Student and Citizen Identities Linked

D3.1 Technical documentation on common code of Platform

Document Identification			
Status	Final	**Due Date**	29/05/2020
Version	1.0	**Submission Date**	03/06/2020

Related Activity	Act 3	**Document Reference**	D3.1
Related Deliverable(s)	D2.1, D3.2, D3.3	**Dissemination Level**	PU
Lead Organization	Atos	**Lead Author**	Ross Little (Atos)
Contributors	ATOS, UJI, UAegean, UPorto	**Reviewers**	GRNET
			UPORTO

Keywords:
Federation, Blockchain, Self-sovereign Identity, Identity, Attributes, eIDAS, EduGAIN, Cloud, Distributed Storage, Verifiable Claims

This document is issued within the frame and for the purpose of the SEAL project. This project has received funding from the European Union's Innovation and Networks Executive Agency – Connecting Europe Facility (CEF) under Grant AGREEMENT No INEA/CEF/ICT/A2018/1633170; Action No 2018-EU-IA-0024. The opinions expressed, and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Aragó, Francisco José	UJI
Triantafyllou, Nikos	UAEGEAN
Pereira Bruno	UPorto
Viellegas, Miryam	Atos
Cortes, Raquel	Atos
Buendia, Carlos	Atos

Document History			
Version	Date	Change editors	Changes
0.1	11/05/2020	Ross Little	First proposal of ToC
0.2	21/05/2020	Ross Little	Updated for user story, use case sequence diagrams, deployment reference architecture.
0.3	22/05/2020	Francisco Aragó	Developed content of sections 4.1, 4.2, 4.4, 4.5.
0.4	22/05/2020	Miryam Villegas	Added in API GW in section 5.4
0.5	22/05/2020	Ross Little	Updates to Section 4 and 5 from UAEGEAN related to VC Issuer and SSI Blockchain descriptions. Updates to section 4 by UPORTO for the persistence module. Updates to exec summary and Introduction and section 5 by Atos Final version to be submitted for review.
0.6	25/05/2020	Francisco Peixoto	Small corrections in review of deliverable.
0.7	29/05/2020	Ross Little	Updated for review comments by GRNET.
0.8	29/05/2020	Ross Little	Updated for links to sequence diagrams.
1.0	03/06/2020	ATOS	Quality review, submission to EC

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Ross Little (Atos)	29/05/2020
Technical manager	Francisco Aragó (UJI)	29/05/2020
Quality manager	ATOS	01/06/2020
Project Manager	ATOS	01/06/2020

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AP	Attribute Provider
AuthN	Authentication
CEF	Connecting Europa Facility
DID	Decentralised Identifier
DSA	Domain Specific Attributes
Dx.y	Deliverable number y belonging to WP x
eIDAS	Electronic Identification Authentication and trust Services
eMRTD	Electronic Machine Readable Travel Documents
EU	European Union
GUI	Graphical User Interface
HEI	Higher Education Institution
IdP	Identity Provider
IF	Interface
KYC	Know Your Costumer
LoA	Level of Assurance
ms	microservice
MS	Member State
PDS	Personal Data Store
OIDC	OpenID Connect
SAML	Security Assertion Markup Language.
SEAL	Student and Citizen Identity Linked
SP	Service Provider
SSI	Self-Sovereign Identity
UA	User Agent
VC	Verifiable Credential

1 Executive Summary

The audience of this document is primarily for the persons interested to deploy SEAL and contribute to its further development.

SEAL is an Open Source Project, Co-financed by the Connecting Europe Facility (CEF) of the European Union, and all development modules are to be found on the Git Hub repo: https://github.com/EC-SEAL.

The SEAL Project participates in the European Student Card initiative with the aim to help provide technical means for facilitating modern online experience for Erasmus and student mobility. In particular, SEAL enables students to authenticate online in a trusted way to university and student services with their existing student credentials (in the eduGAIN federation) and/or with their national citizen eIDs (in the eIDAS network). The means on how SEAL achieves this is through providing students with trusted linked identities, that they can share as required with the Service Providers that need them. SEAL supports both a federated and Self-Sovereign Identity solution for Service Provider verifications.

The SEAL service does not store any personal data in a central repository and therefore avoids honeypot for attacks, to steal user´s data, and in the same design addresses some MS´s privacy concerns in creating central databases, with such linked information. All student identity and linked identity data is stored in the students´ own Personal Data Store and/or SSI Wallet, as they prefer.

It is seen in the presented design, that SEAL is developed in an extendable microservice architecture which makes it easy to add new modules for example for importing identities beyond those sources currently supported (i.e. eduGAIN, eIDAS & eMRTD).

The document first presents typical user stories that SEAL helps to facilitate, before going on to describe the developed logical architecture that is designed, considering also requirements from the grant agreement as outlined in the previous design deliverable [1].

Finally, the document presents the developed microservice solution, detailing the deployment reference architecture, the OpenAPI interface specification and the technical information for each developed microservice module, so to be useful to anyone wanting to later contribute to its coding, and also deploy and test it.

2 Introduction

2.1 Purpose of the document

The document serves to provide an overall description of the SEAL Service from the user stories that it addresses, to the logical architecture and design and finally the developed modules and interfaces.

The document with its references to the project GitHub repo serve as a resource to anyone that wishes to further develop and deploy SEAL.

Any updates to the technical design or deployment reference architecture during integration and test will be captured in a revised final version of this deliverable.

2.2 Relation to other project work

The description of the identity modules that the SEAL Service developed and deploys are described in SEAL D3.2 [2].

The description of the mobile and web Dashboard that the SEAL Service developed and deploys are described in SEAL deliverable D3.3 [3].

Some reference will be made to the above modules and their functions in this deliverable as they are part of the whole SEAL Service deployment, however these documents must be referred to in their own right for a proper examination of their functionality and implementations.

2.3 Structure of the document

This document is structured in 3 major chapters

Chapter 3 presents the user stories for which SEAL aims to provide for

Chapter 4 presents the up-to-date developed logical design architecture

Chapter 5 presents the technical solution, including deployment reference architecture, interfaces, and deployed modules

3 User Story

SEAL aims to facilitate student mobility in Europe by providing students with the means to be able to authenticate to university and other student services with either their national citizen and/or academic eIDs. Wherever students travel to in Europe they will be able to be prove they are a student and identify themselves through their academic and/or citizen identities.

Here are typical student user stories that SEAL aims to provide for:

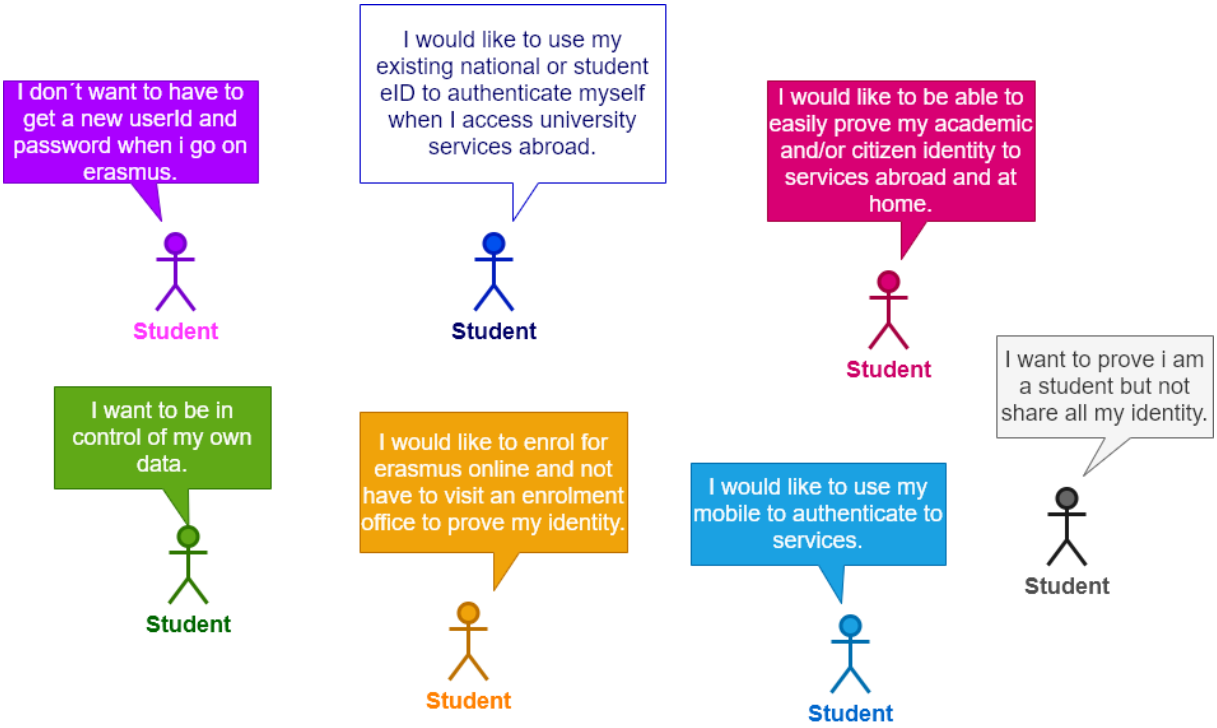


Figure 1 SEAL Student User Stories

4 Logical Design

This section presents the modular modelling of the basic functional units and the design of the interfaces that articulate the behaviour and collaboration between them. Each module can implement more than one interface.

4.1 Use Case Diagrams

This section shows the UML use case diagrams that are designed for the SEAL Service.

4.1.1 SEAL Service Dashboard operations

This use case diagram shows all the operation flows the user can follow from the access to the SEAL Service Dashboard. We will specify separately only those requiring more detail than what is shown here. For the application logic detail of each operation in this diagram, refer to the specific activity diagrams in section 4.4.

Figure 2: SEAL Service Dashboard operations

4.1.2 User requests validated linking between two imported identities by a human agent

This diagram refers to a specific implementation of a linking module, which involves the interaction of another human agent, and reflects the most complex case supported by the linking interface.

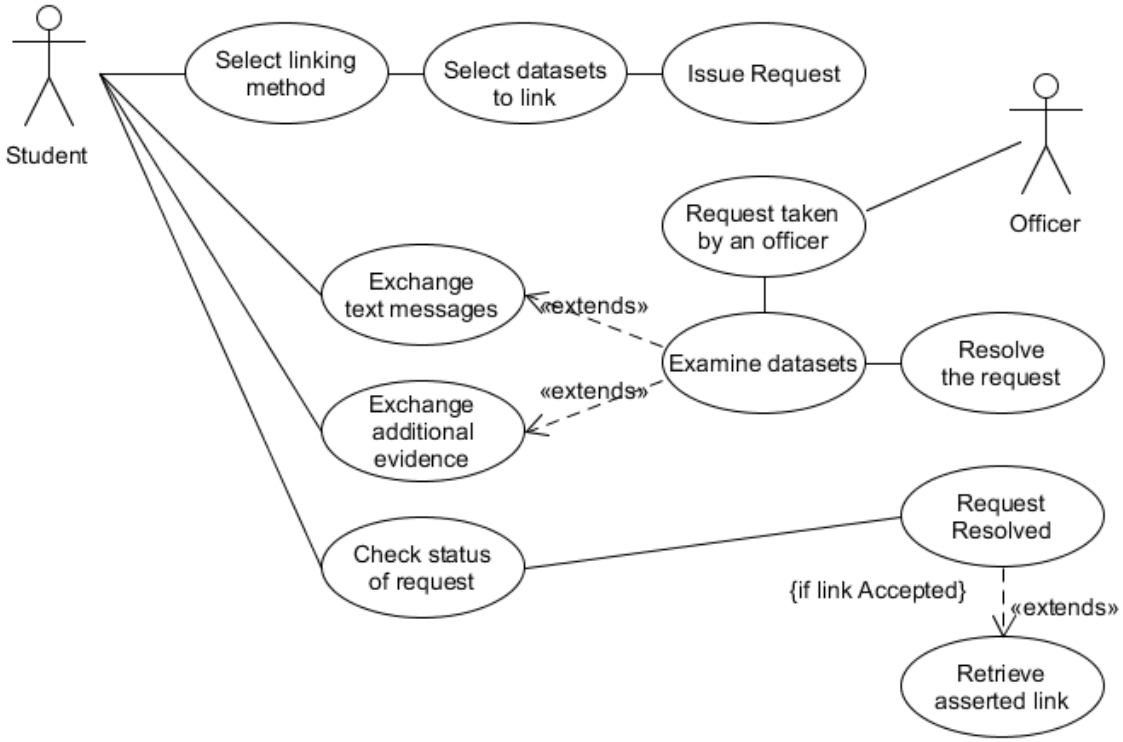


Figure 3: Human-agent identity linking use case

4.1.3 Service Provider requests to authenticate a user and/or verify his/her linked identity attributes

This use case contemplates both the case of federated SPs requesting authentication or data (identity or linking data) to SEAL service, as well as the relay of the SP on a VC verifier.

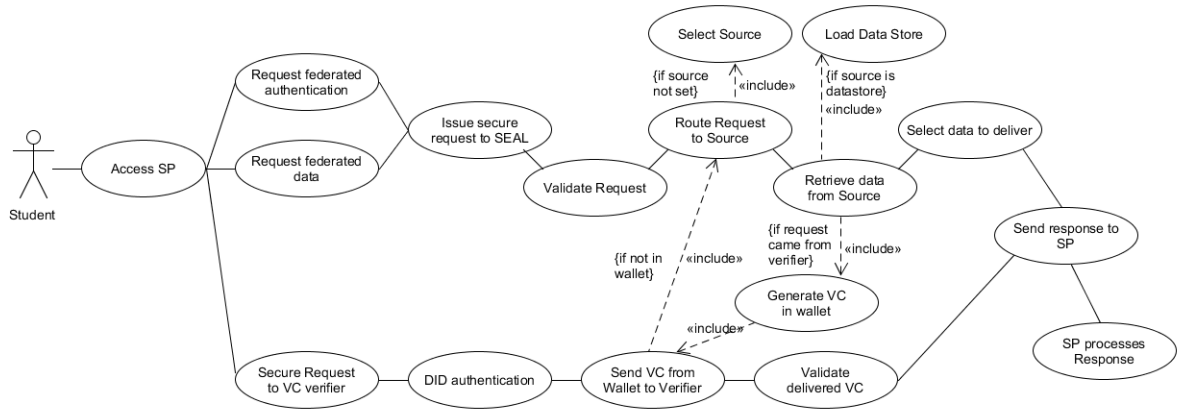


Figure 4: Service Provider request use case

4.2 SEAL Logical Architecture

4.2.1 SEAL Logical Architecture Diagram

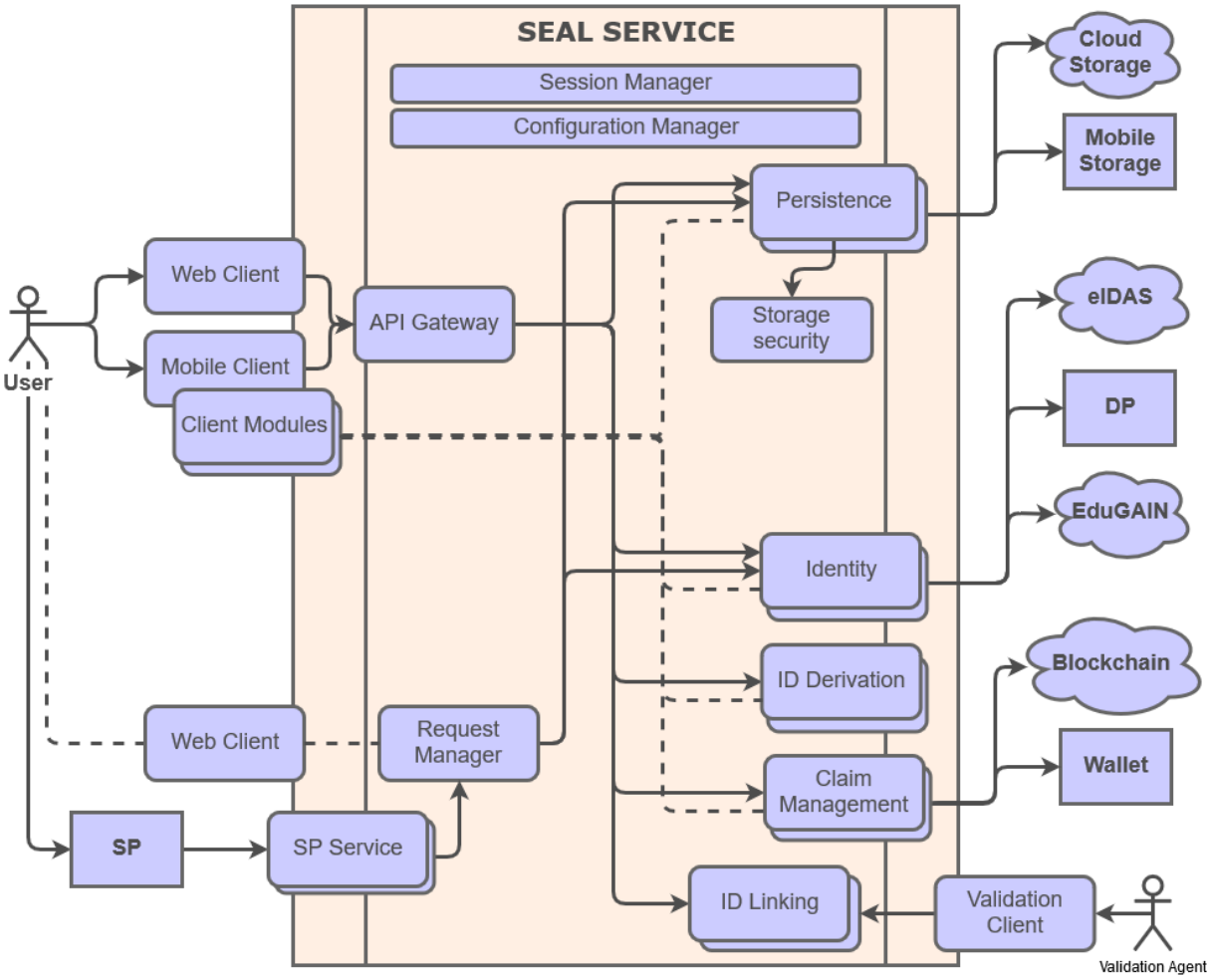


Figure 5: SEAL Service architecture diagram

4.2.2 Modules Definition

Textual definition of the modules and their purpose and general functionality include sequence or activity diagrams where needed:

4.2.2.1 Session

To keep the user operation state between microservice calls. Designed as the session layer of any application framework. Acts as a secure volatile shared buffer between microservices to exchange data in the context of a client session.

Also, it implements operations to facilitate the secure communication between front-channel relayed microservice calls.

4.2.2.2 Configuration

Module to keep a central repository of configuration, both private and public configuration, to avoid multiplicity.

4.2.2.3 API GW

This module implements the user interface API of the SEAL service, and thus implements the main application and control logic of the service. This allows supporting multiple independent clients, like the Web app and the mobile client.

This module relays basically presents the selection lists and display data to the client and brokers the command requests from the client to access specific modular functionalities.

4.2.2.4 Client

These modules implement the common part of the user interface (The SEAL Dashboard) for the web browser environment and for the mobile application environment. They are considered not trusted running environments, so they simply act as command and display interfaces. The modules that require specific interface handle this by themselves. To allow for this, the clients must have the option to pass control to a remote service UI (through a browser window, iframe or equivalent mechanism).

4.2.2.5 Identity

This module will allow trusted access to information regarding an identity, and if available, authentication with that identity. EduGAIN, eIDAS and eMRTD are specific implementations of this module. Implementations that use front-channel redirection to external sources support an external call-back interface, so to receive the response.

4.2.2.6 Persistence

Persistence modules are those where application data will be encrypted and decrypted for storage. Implementations include personal cloud storage services (Google Drive and Microsoft One Cloud), SEAL mobile app storage and Browser local storage.

In SEAL, the Personal Data Store (PDS) to be stored will be encrypted with a user password, to disallow unauthorised access, and signed by the SEAL service, to prevent tampering.

When the PDS is retrieved from its local or cloud storage, it will be loaded into the user´s SEAL session, so it can be managed or updated by the dashboard operations.

4.2.2.7 Identity Derivation

These modules allow to generate a new identifier, linked to the data sets of the user, with a high assurance level. Depending on the implementation, it might also produce credentials to authenticate. A random UUIDv4 generation module is provided.

4.2.2.8 Identity Linking / Reconciliation

Identity linking modules receive two data sets related to two identities and implement a procedure to check if both identities belong to a same person. These procedures may be:

- Automated: the procedure does not imply any human agent as trusted third-party validator.
- Semi-automated: the procedure implies a human agent as trusted third-party validator but does not require the user to identify himself in person in front of the validator.
- Manual: the procedure implies a human agent as trusted third-party validator and requires the user to identify himself in person in front of the validator.

Two implementations are provided:

- An automated linking module, that relays on a configurable ruleset to transform incoming data sets into a set of strings that are semantically paired and are then compared with a string similarity algorithm, and based on the results and a configured threshold, the link is awarded or not.
- A manual linking module, that has its own integrate client for the linking officers to handle the requests, examine their content, interact with the requester

Specification is ready to support third party KYC companies. Provided modules do not store personal user data (except in session, during the reduced handling period) to avoid GDPR issues. Any third-party integrator must implement a consent form for the user to accept the data treatment and transfer to the external provider.

4.2.2.9 SP request handling

These services implement all the public interfaces to provide functionalities to client services (SPs). These include authentication, attribute and linking information query, validation of claims, and possibly others. A SAML and an OIDC modules are provided.

4.2.2.10 Verifiable Credentials Issuer

This module will allow issuing/deleting/revoking/refreshing/updating verifiable claims, which are data sets that have validity and integrity on their own or through a deferred validation mechanism, so the user can deliver them to an SP who will perform the validation by itself. SEAL Service, of course, might need to write the verification means on a specific infrastructure, and the claim on a persistence module. Issuing verifiable credentials on a DID wallet and writing the verification on a Blockchain is the main implementation of this module, but it could also be implemented on a PKI infrastructure.

4.3 Sequence Diagrams

4.3.1 User accesses SEAL Service

This use case concerns the user accessing the SEAL Service through the Mobile or Web Dashboard App. There are two options available to access the SEAL Service: Either by:

- loading his/her PDS into SEAL and proving they own it by deciphering it with a secret password.
- a user authenticating to SEAL with their Self-Sovereign Identity Wallet.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC1

4.3.2 User configures a PDS

This use case concerns the user configuring their Personal Data Store (PDS), be it on a mobile device, a local browser or in a cloud service such as Google Drive or OneDrive.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC2

4.3.3 User adds/imports one of their identities

This use case concerns the capability of SEAL to add / import electronic identities from authentication sources and other attribute sources. Currently supported are eIDAS, eduGAIN and eMRTD.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC3

4.3.4 User wants to move/backup data to another PDS

This use case concerns the capability of SEAL to let a user move/change his PDS from one storage medium to another e.g. from mobile storage to cloud storage.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC4

4.3.5 User requests to be issued with a Verifiable Credential

This use case concerns the capability of SEAL to issue a user with a Verifiable Credential for any of their identities added in SEAL.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC5

4.3.6 User requests a derived identity

This use case concerns the capability of SEAL to create derived identities linked to a previously authenticated identity. For example, anonymous UUID, over18, isStudent etc.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC6

4.3.7 User requests validated linking between two imported identities

This use case concerns a user who requests that two of his/her identities are validated by SEAL or an external 3rd party to the fact that they are linked/belong to the same user, and with a certain Linked Level of Assurance (LLoA). There can be manual or automatic means to validate the user´s identities are linked.

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC7

4.3.8 Service Provider requests to authenticate a user and/or verify his/her linked identity attributes

SEAL supports Service Providers with various verification requests for authentication or identity attributes query as follows:

1. Authentication, with option to specify eIDAS or eduGAIN source (by a variable).
2. Retrieval of linked identity attribute data, with option to specify one of the following sources (by a variable):
    - Personal Data Store
    - Verifiable Credentials from a user´s SSI Wallet
    - eIDAS
    - eduGAIN
3. No specific request and user can select from any of the verification means supported by SEAL

Links to the sequence diagrams supported for this use case are available here:

- https://github.com/EC-SEAL/interface-specs/wiki/UC8

4.4 Activity Diagrams

4.4.1 Dashboard access

This is the logic flow when a student accesses SEAL service dashboard from its client. Blue boxes reference to other activity diagrams that expand this functionality

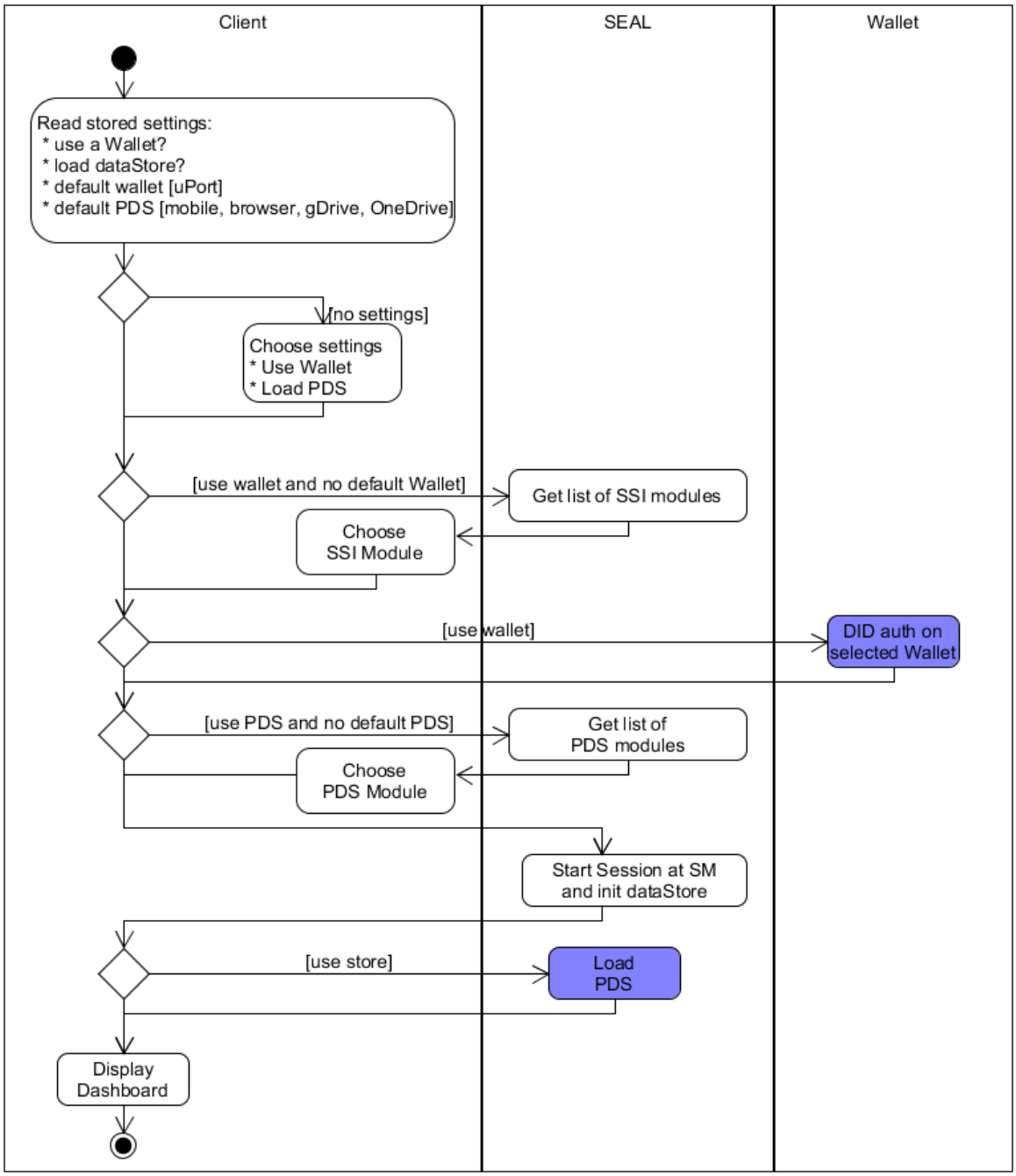


Figure 6: Dashboard Access activity diagram

4.4.2 DID Authentication

Flow of the DID-pairing with the user wallet.

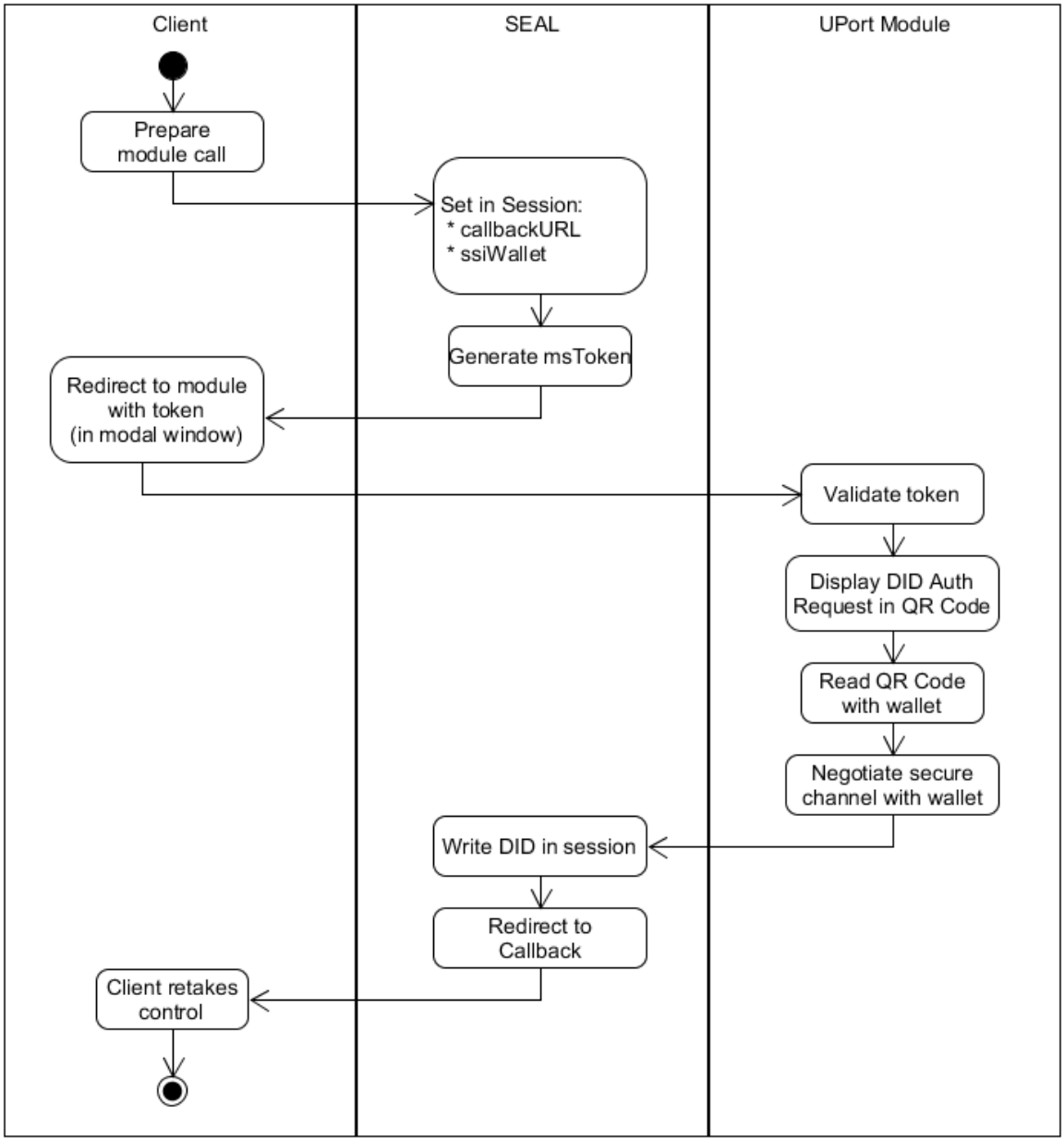


Figure 7: DID Authentication activity diagram

4.4.3 Configuration Manager

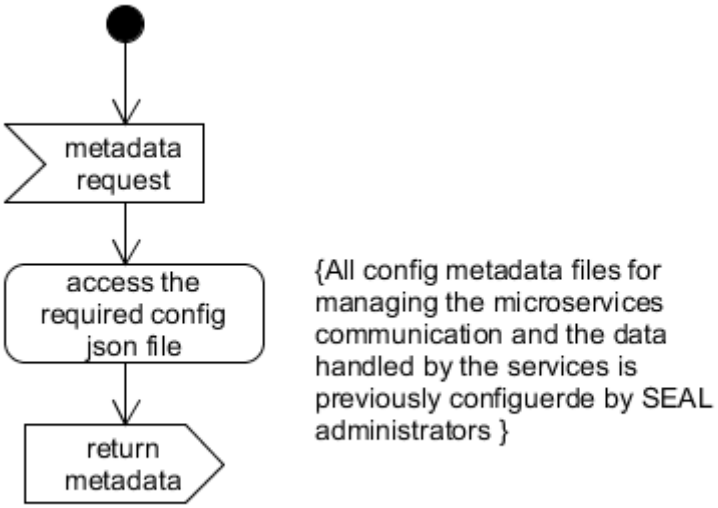


Figure 8 Configuration Manager main business logic

4.4.4 Session Manager

4.4.4.1 Generate msToken

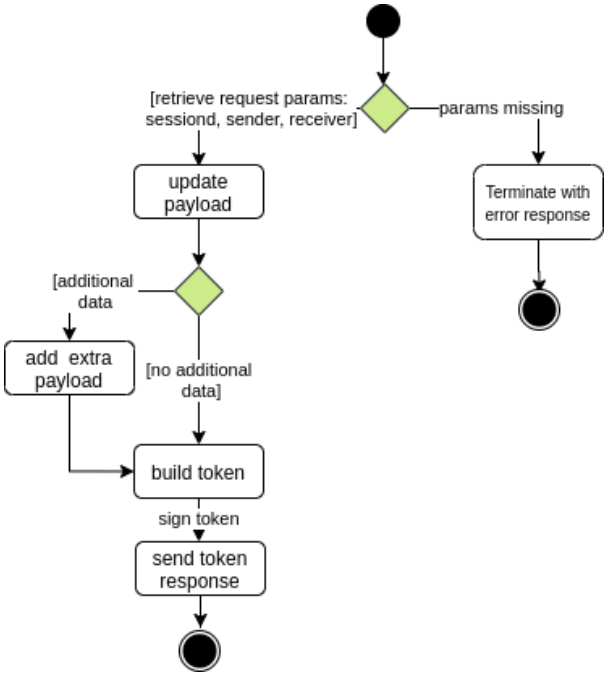


Figure 9 Generate msToken

4.4.4.2 Validate msToken

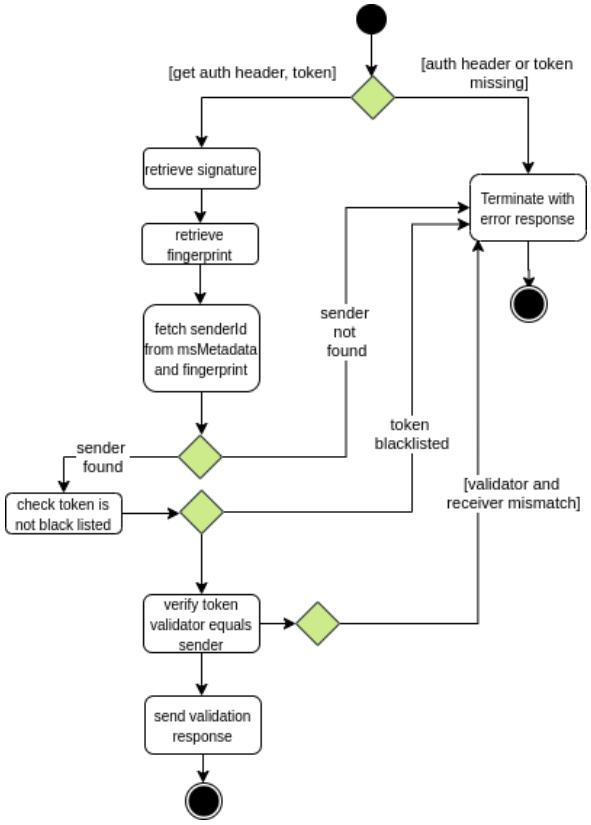


Figure 10 Validate msToken

4.4.4.3 Update Session data

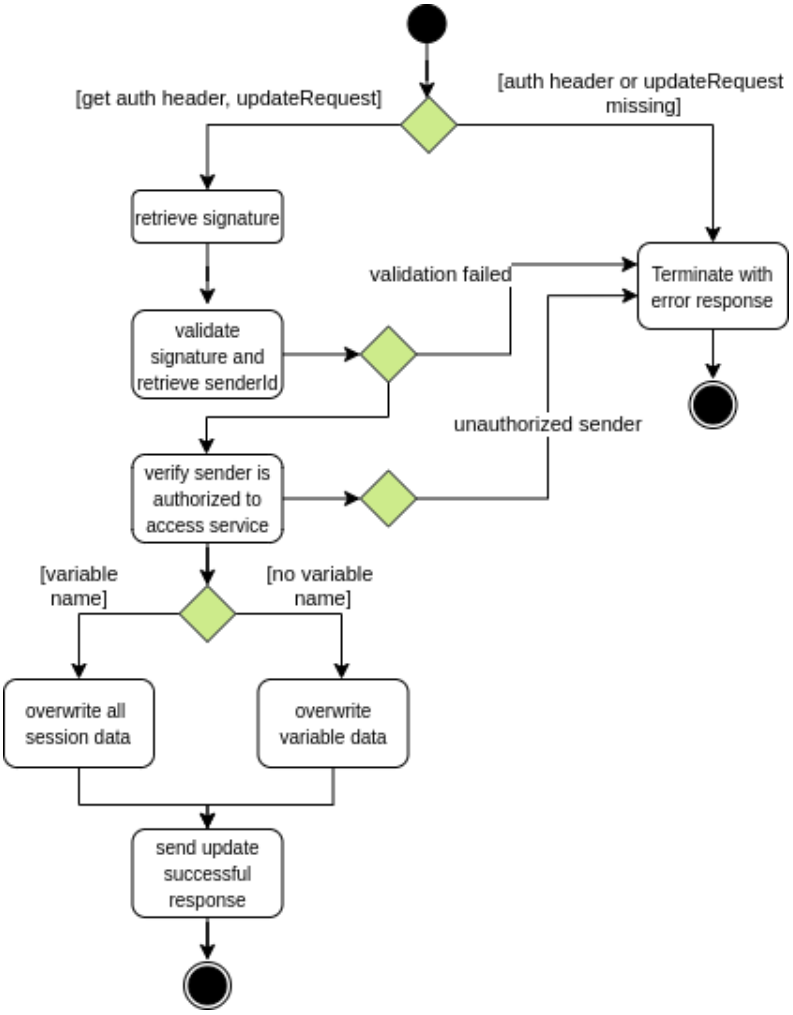


Figure 11 Updates Session Data

4.4.5 API GW

The main role of the API GW is to be the first point of interaction with the Dashboard App user interface. It performs business logic and service orchestration with backend microservice SEAL systems and responds to the Dashboard with the access method and security token needed to communicate with the service that the dashboard is wanting to access. This is captured in the following activity diagram.

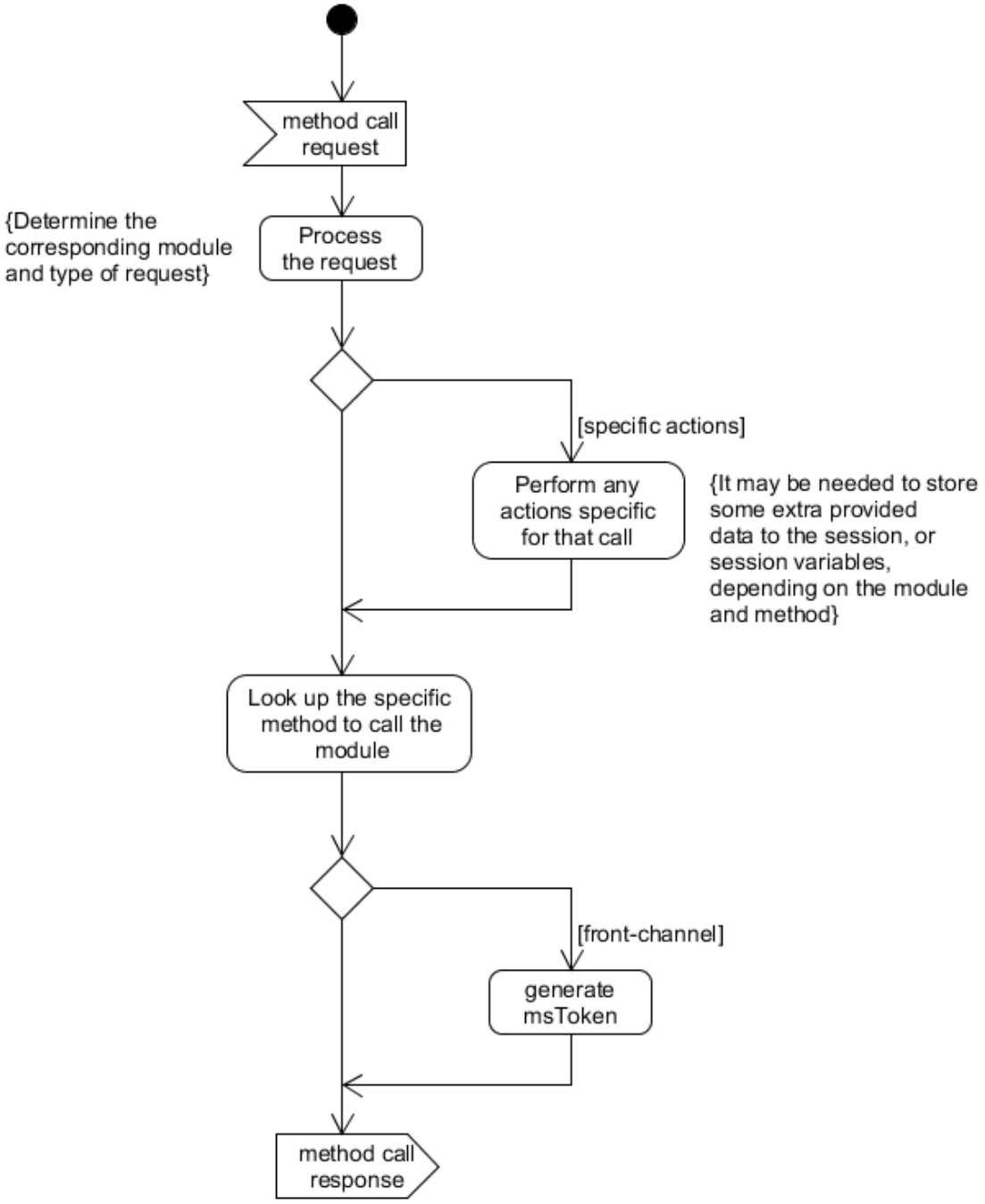


Figure 12 API GW main business logic

4.4.6 Persistence

This section describes the different logical persistence storage management flows. Only the load operation is described, as the write is essentially equivalent, and just reverses the order of the core

operations: (sign, encrypt and write instead of read, decrypt and verify signature). The three generic flows are as follows:

- Remote storage on a cloud service
- Local storage on the same client we are using.
- Storage on the mobile client, but accessing it from the web client

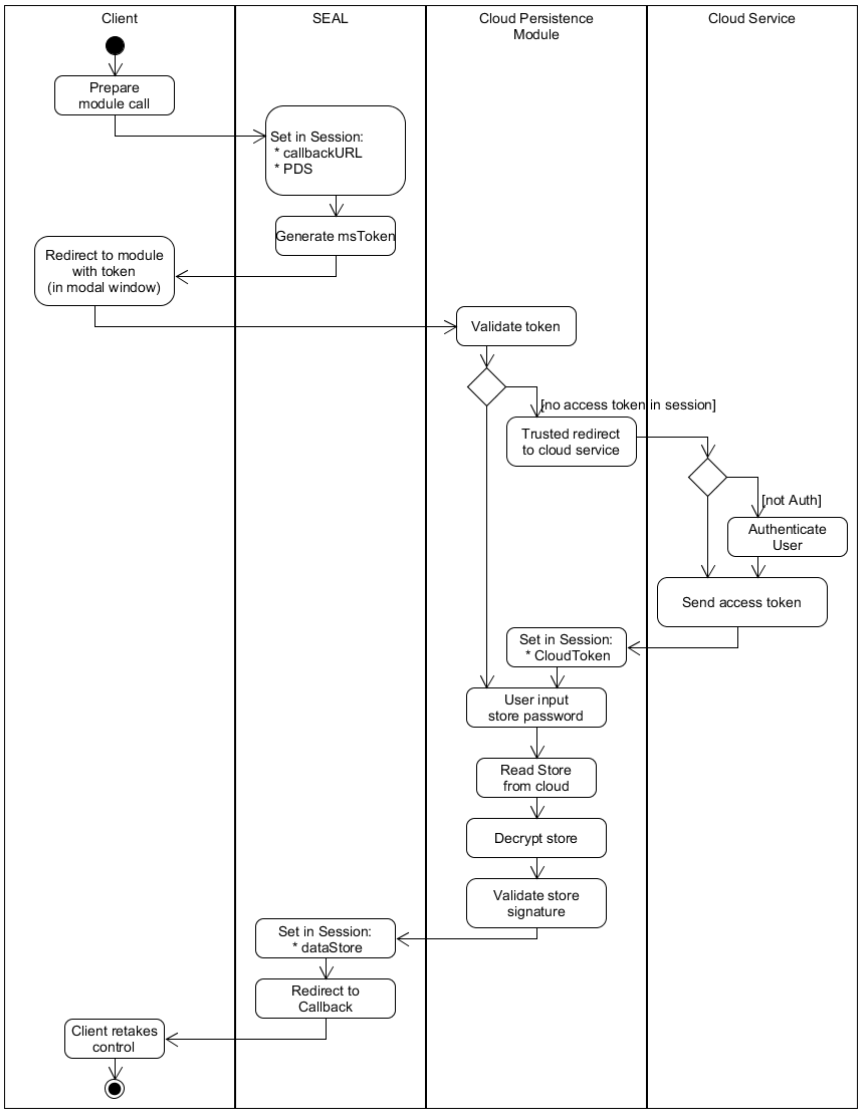


Figure 13: Persistence in cloud activity diagram

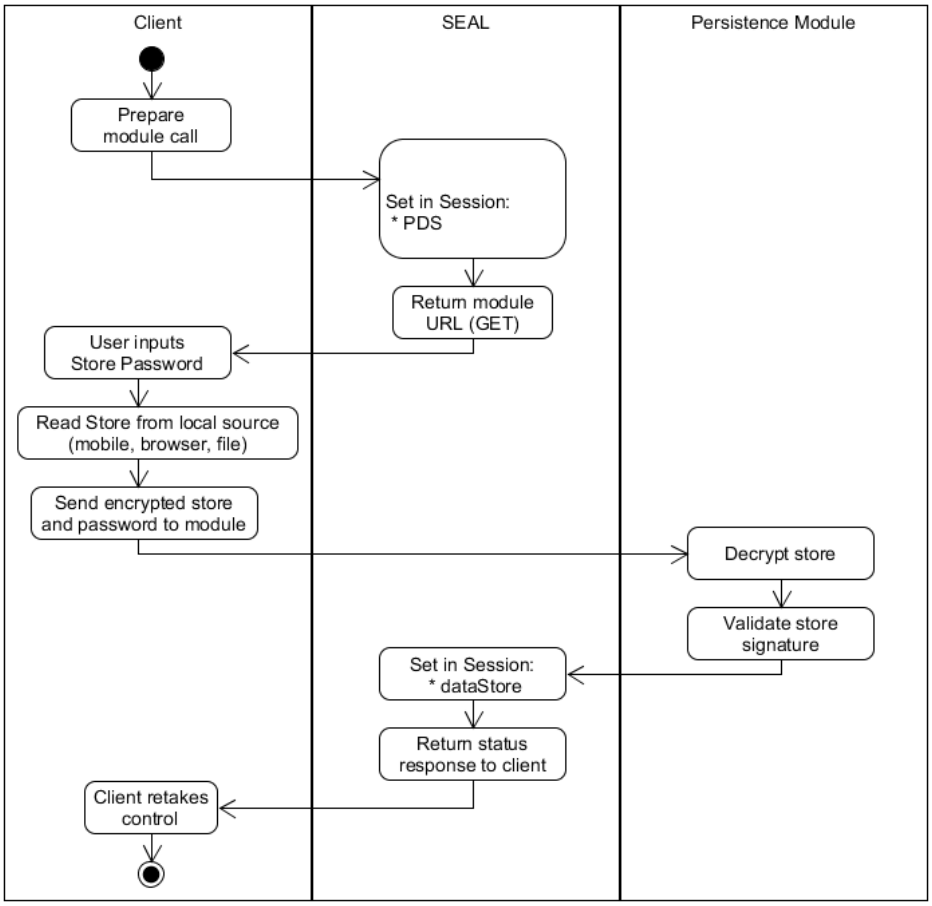


Figure 14: Persistence in local client activity diagram

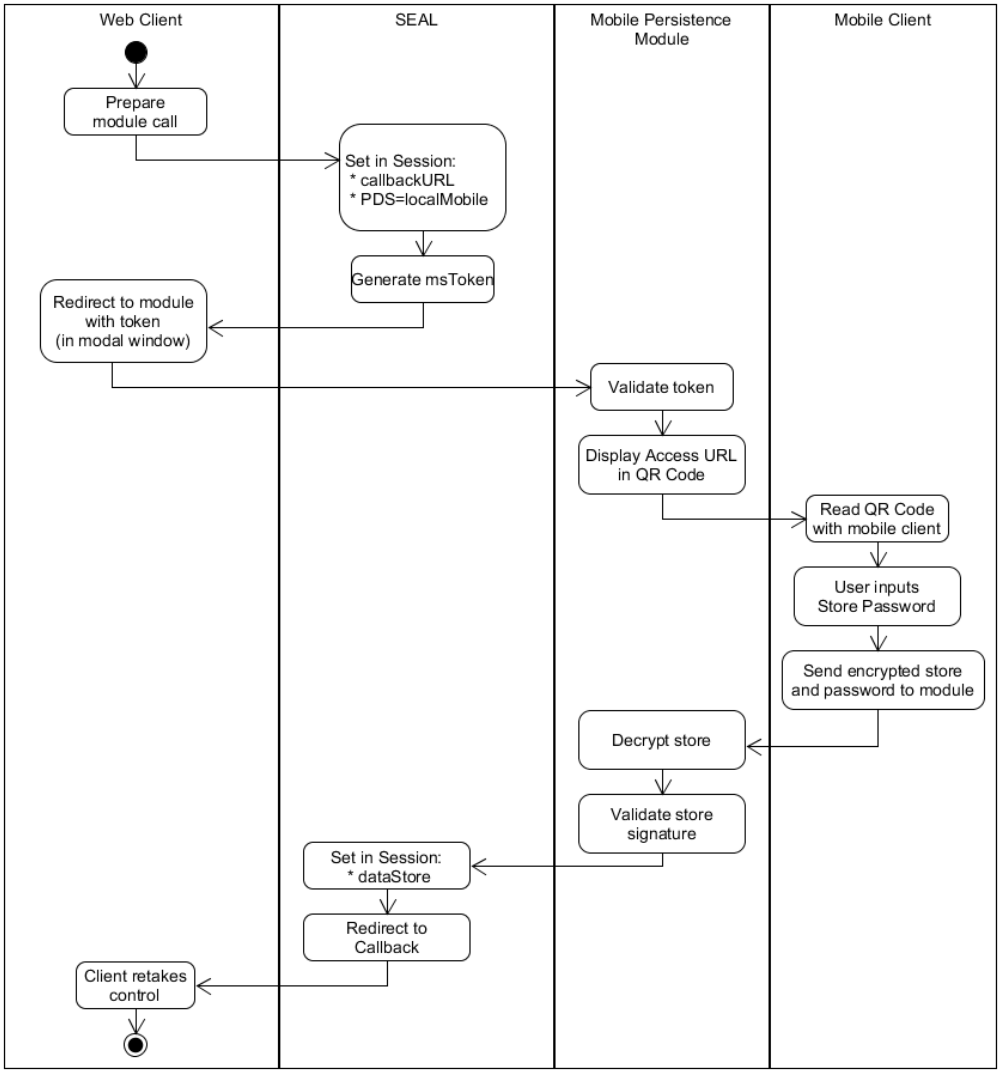


Figure 15: Persistence in remotely accessed mobile client activity diagram

4.4.1 Federated Identity Import/Authentication

Flow for importing the data from a federated identity (eIDAS, EduGAIN, etc.)

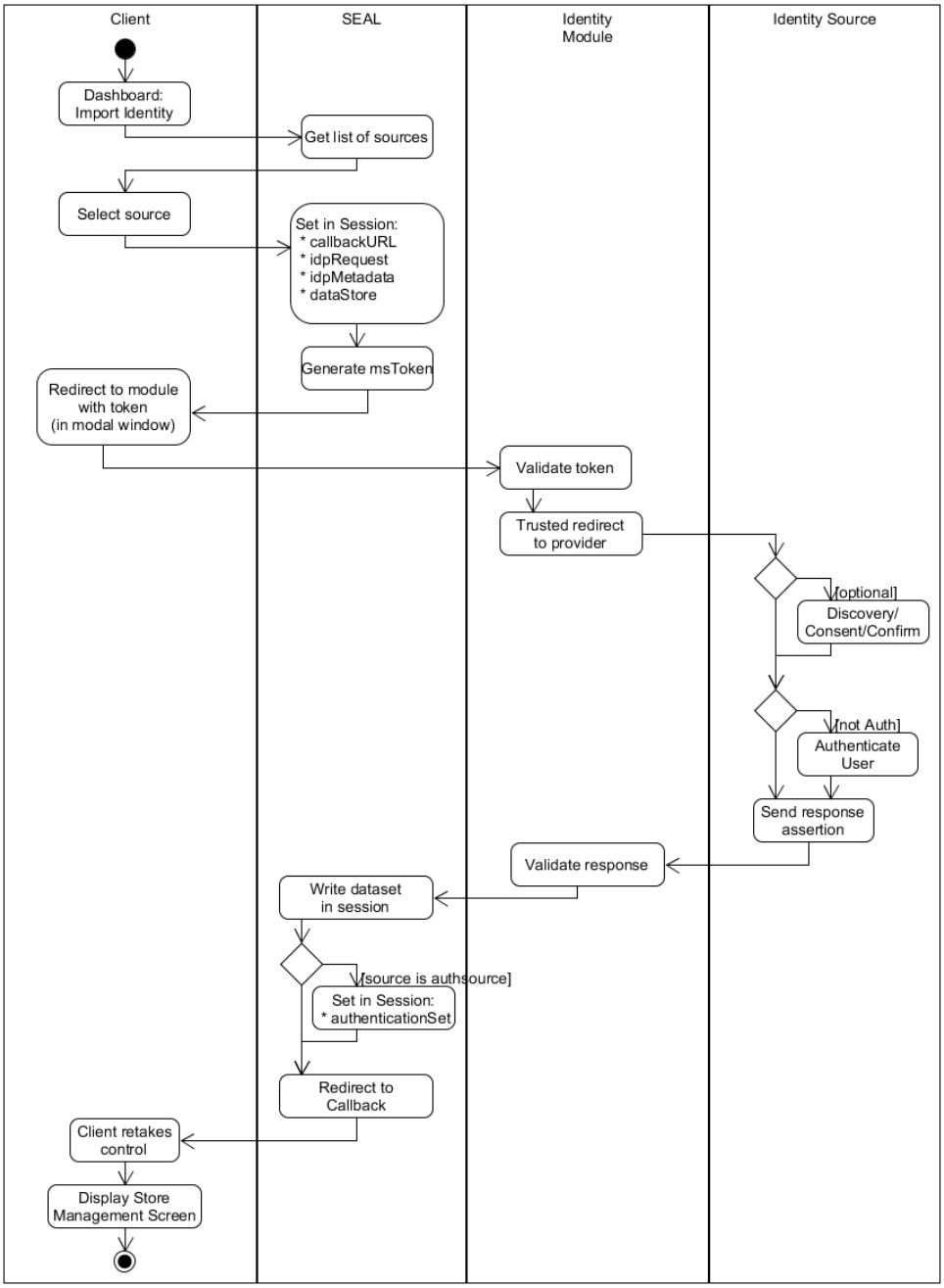


Figure 16: Federated Identity import activity diagram

4.4.2 Travel Document Identity Import

Flow for importing the data from an eMRTD.

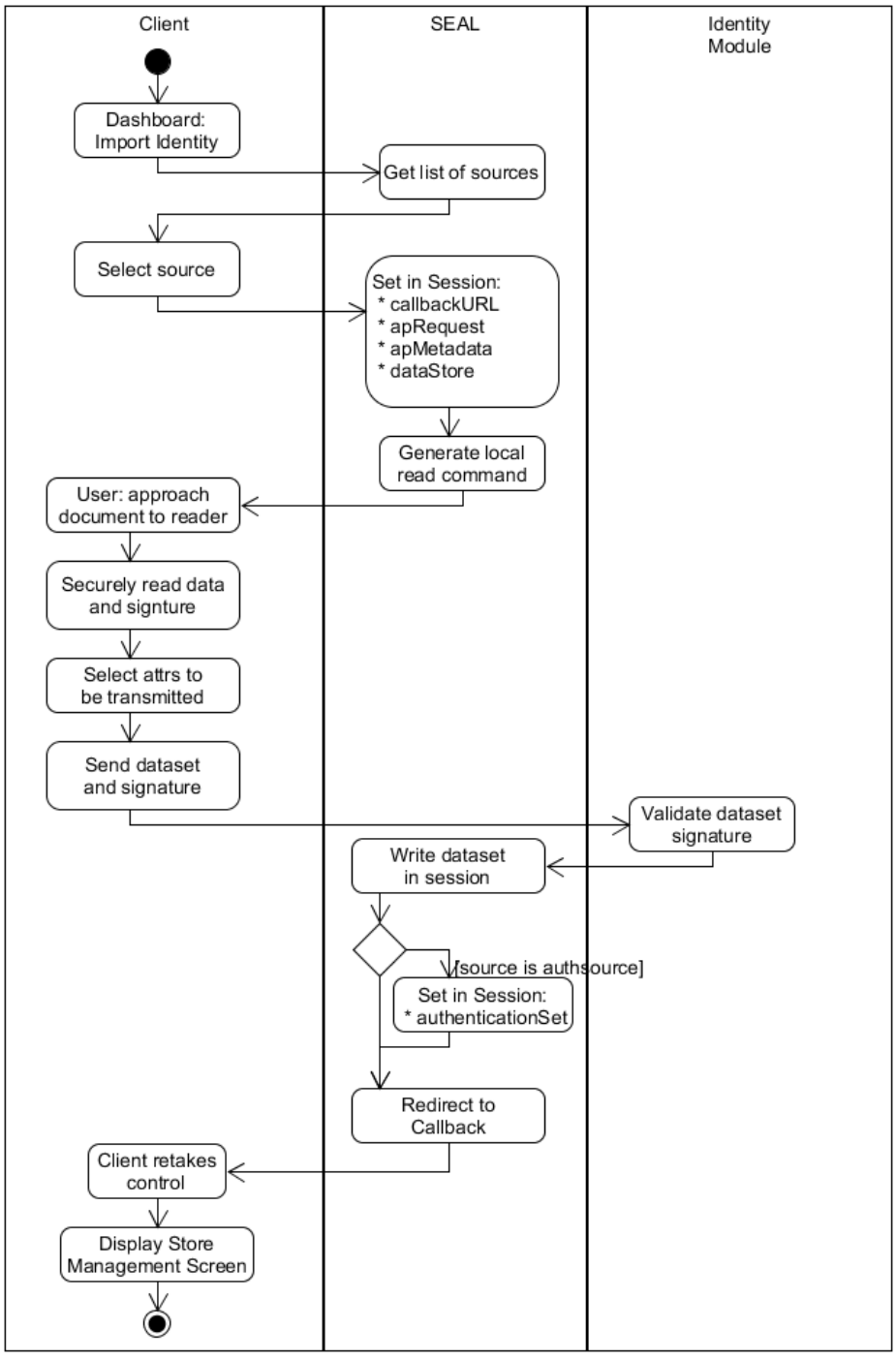


Figure 17: eMRTD identity import activity diagram

4.4.3 Derivation

The derivation module currently supported in SEAL is to derive a simple UUID that can be used for anonymous user survey´s etc.

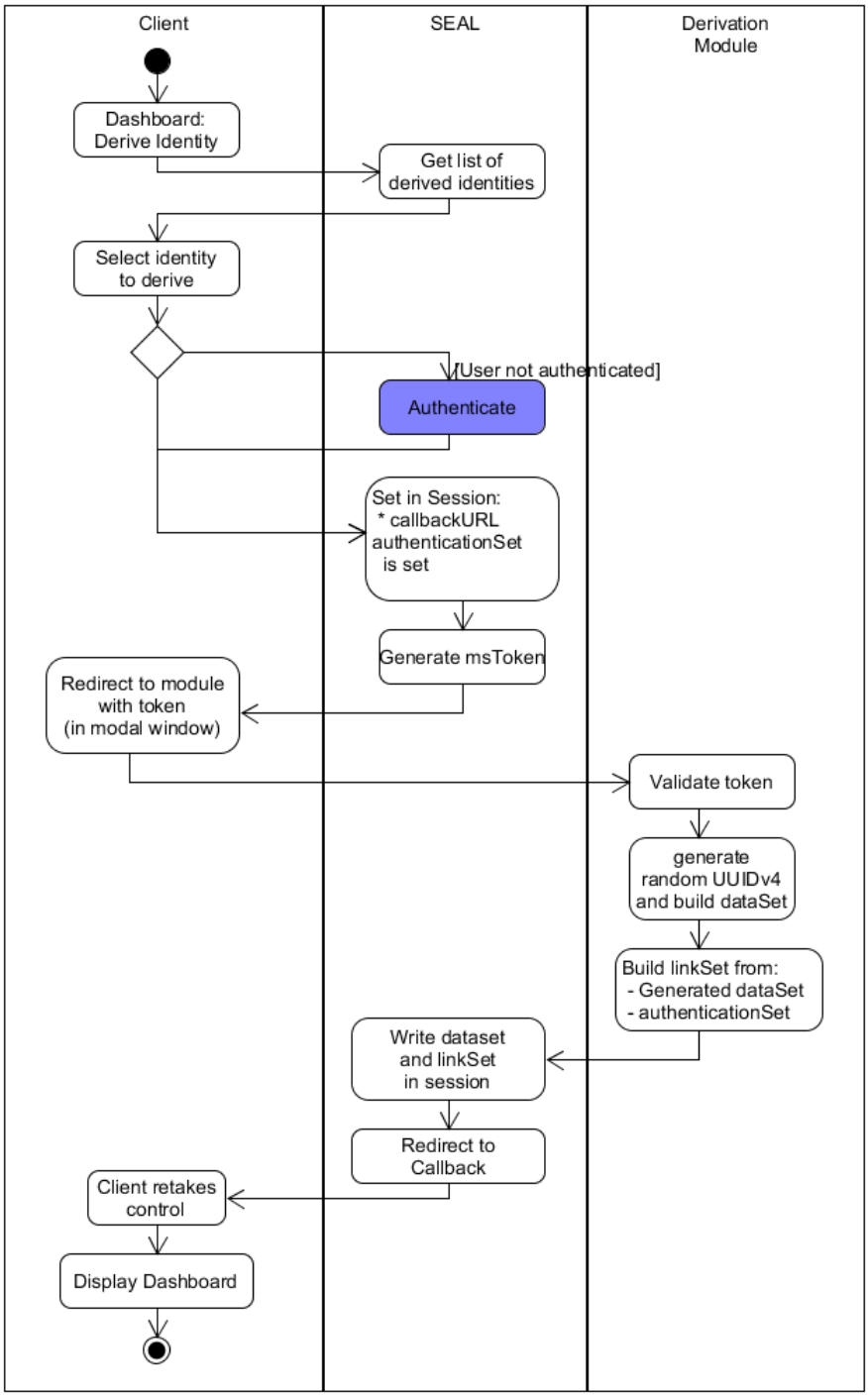


Figure 18 Derive UUID main business logic

4.4.4 Identity Linking / Reconciliation

Here we provide the flow for the two linking modules included in SEAL: automated linking and manual linking.

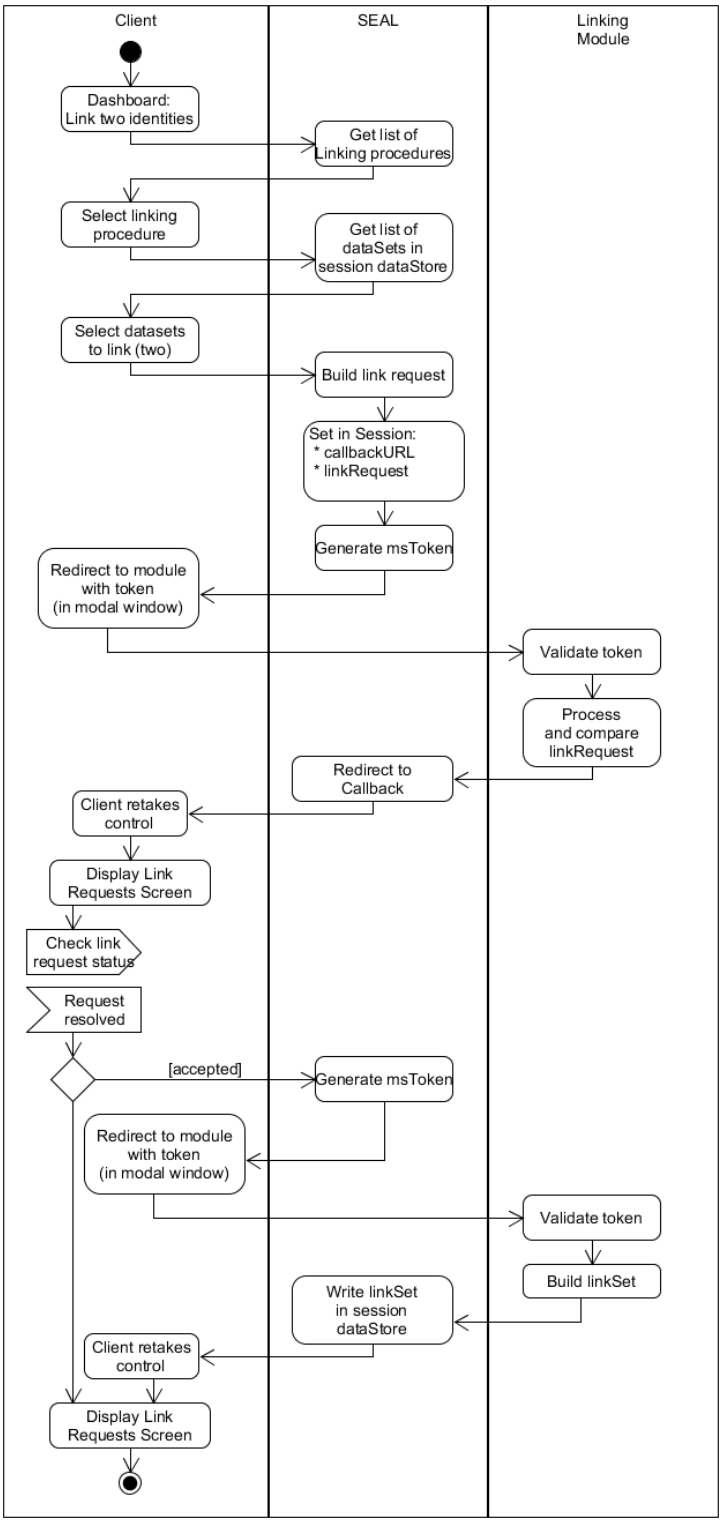


Figure 19: Automated linking flow activity diagram

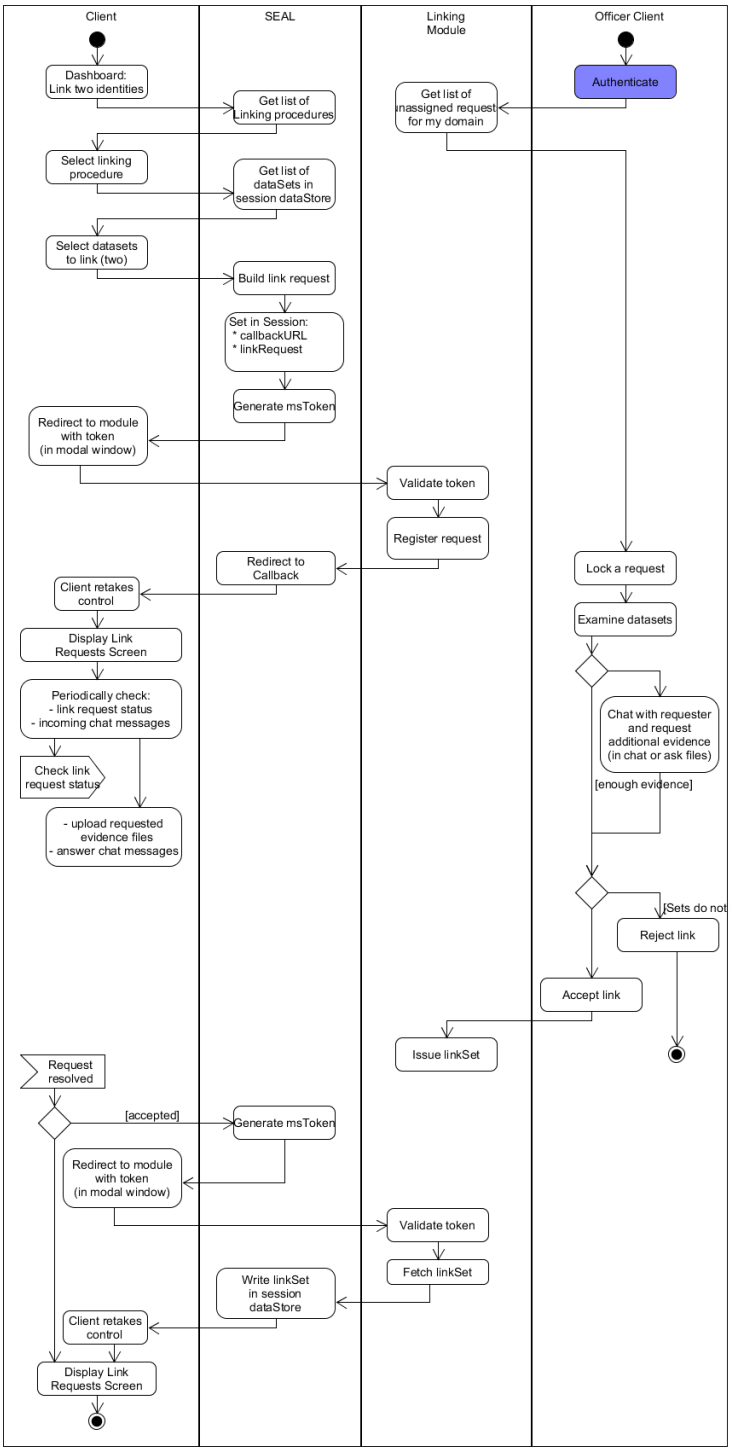


Figure 20: Manual linking flow activity diagram

4.4.5 VC Issuer

4.4.5.1 VC Issuer

The flow for generating a Verifiable Credential from the sources available in SEAL

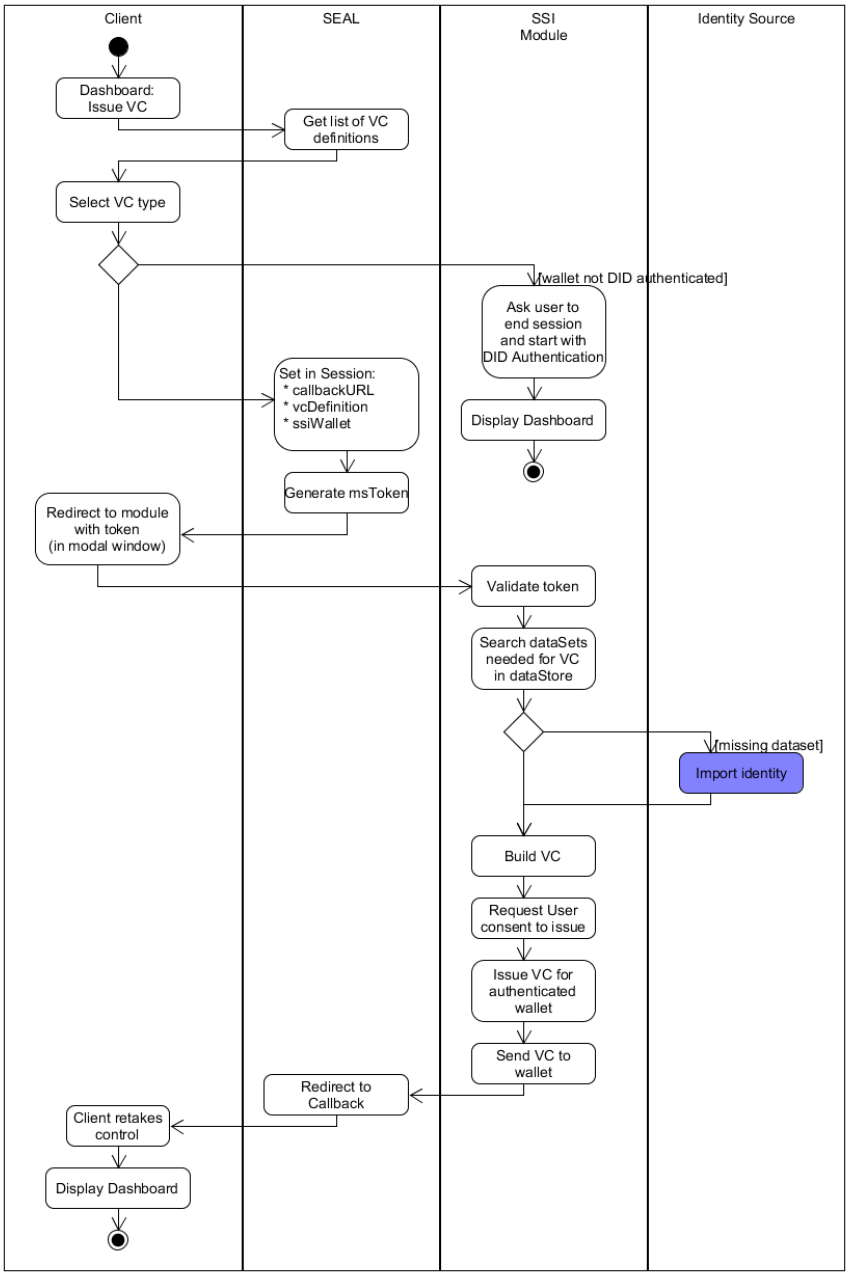


Figure 21: VC issuer activity diagram

4.4.5.2 VC Issue eIDAS activity logic

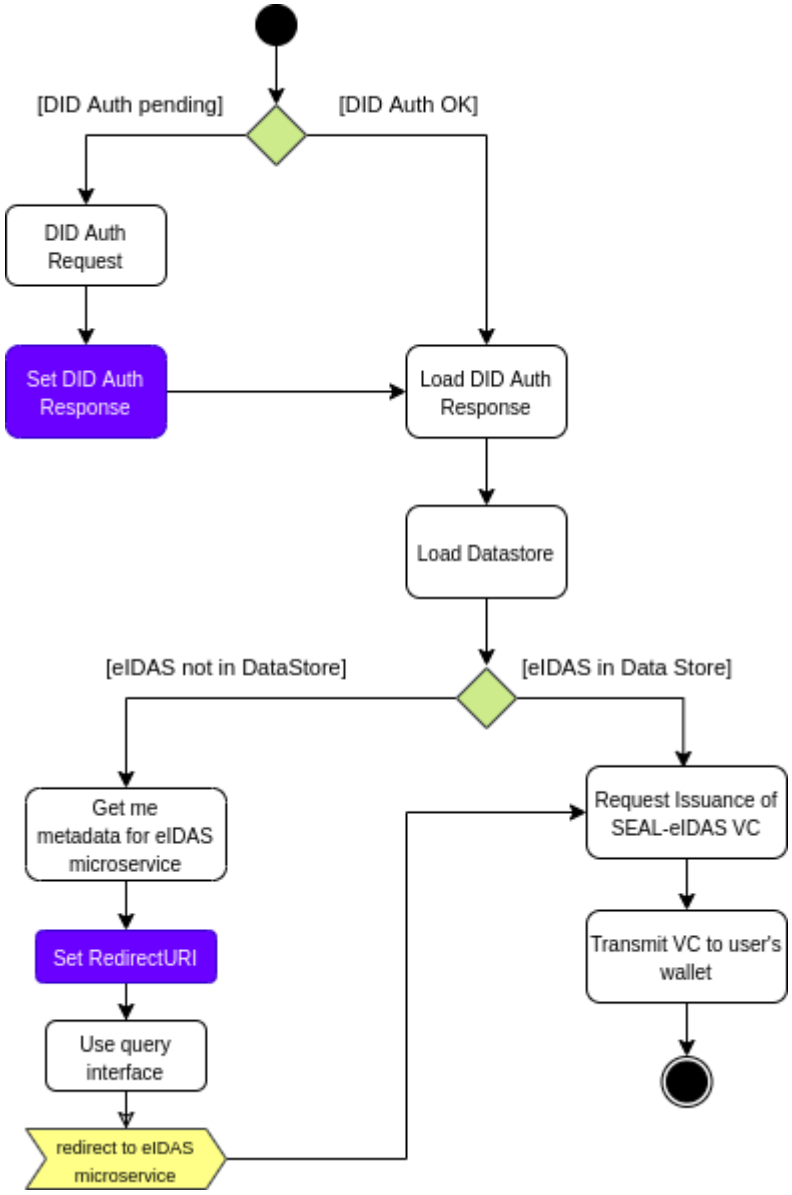


Figure 22 VC Issue eIDAS activity logic

4.4.5.3 VC Issue eduGAIN activity logic

The logic of this flow is identical to the eIDAS one presented in section 4.4.5.2 with the only difference that authentication is redirected to the eduGAIN microservice instead of the eIDAS one.

4.4.5.4 DID Auth activity logic

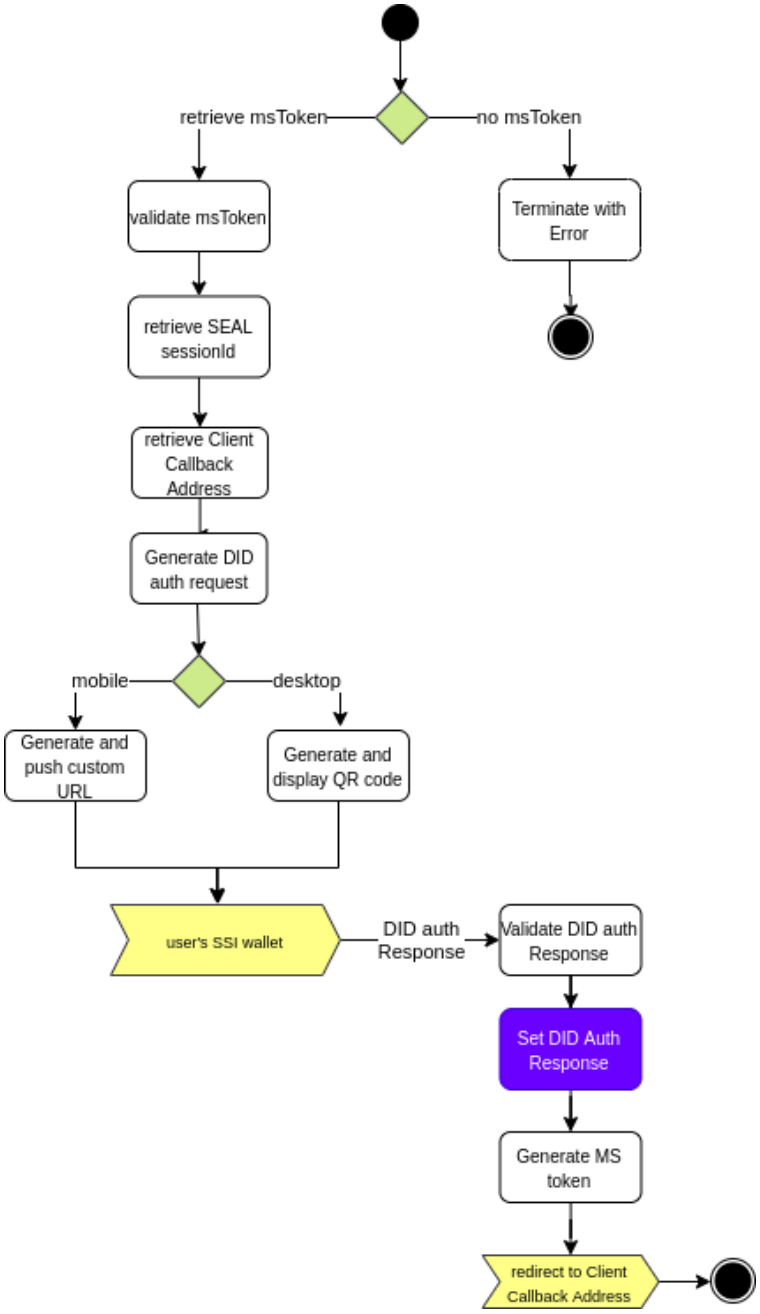


Figure 23 DID Auth activity logic

4.4.6 Request Manager

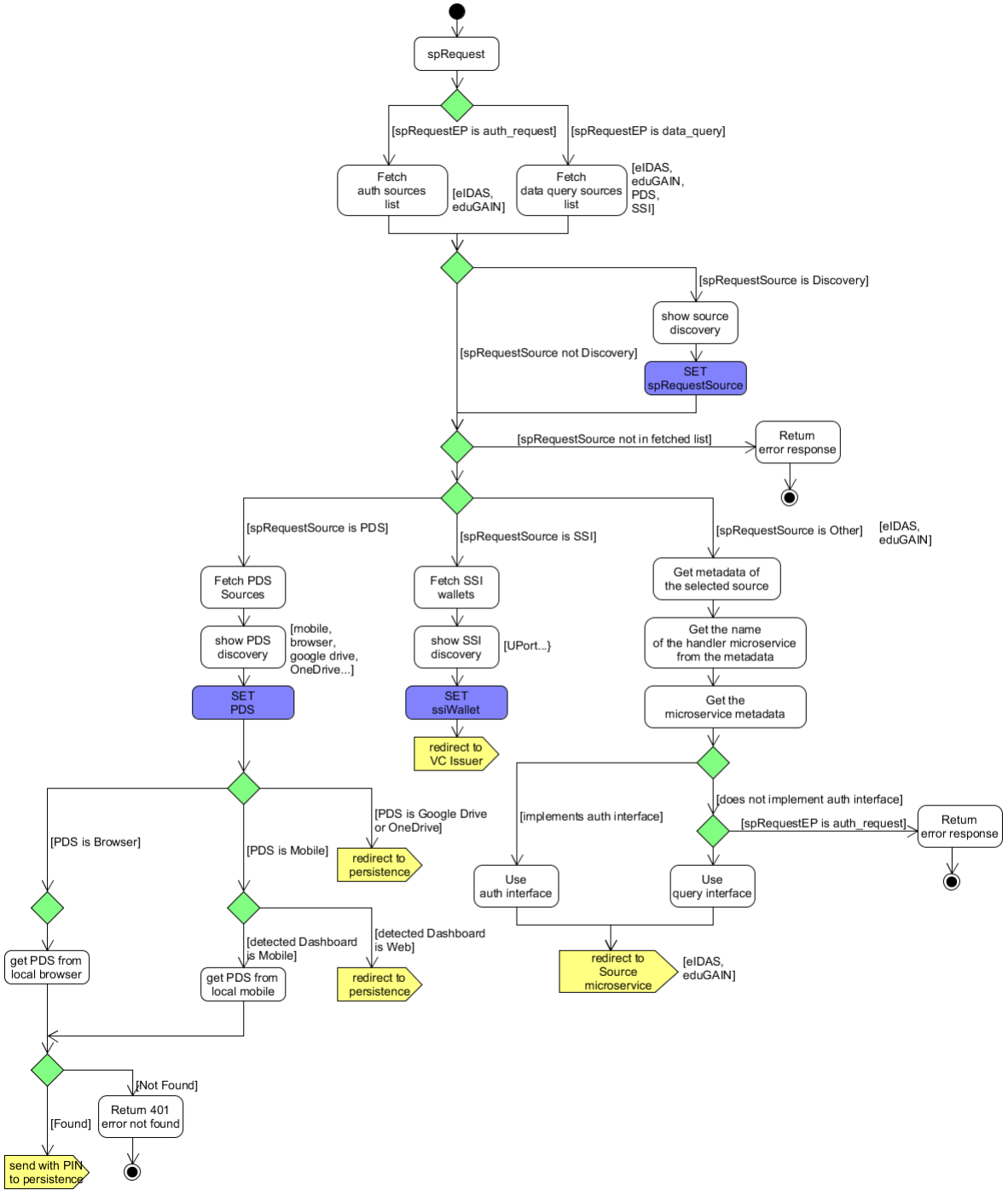


Figure 24 Request Manger SP Request handling

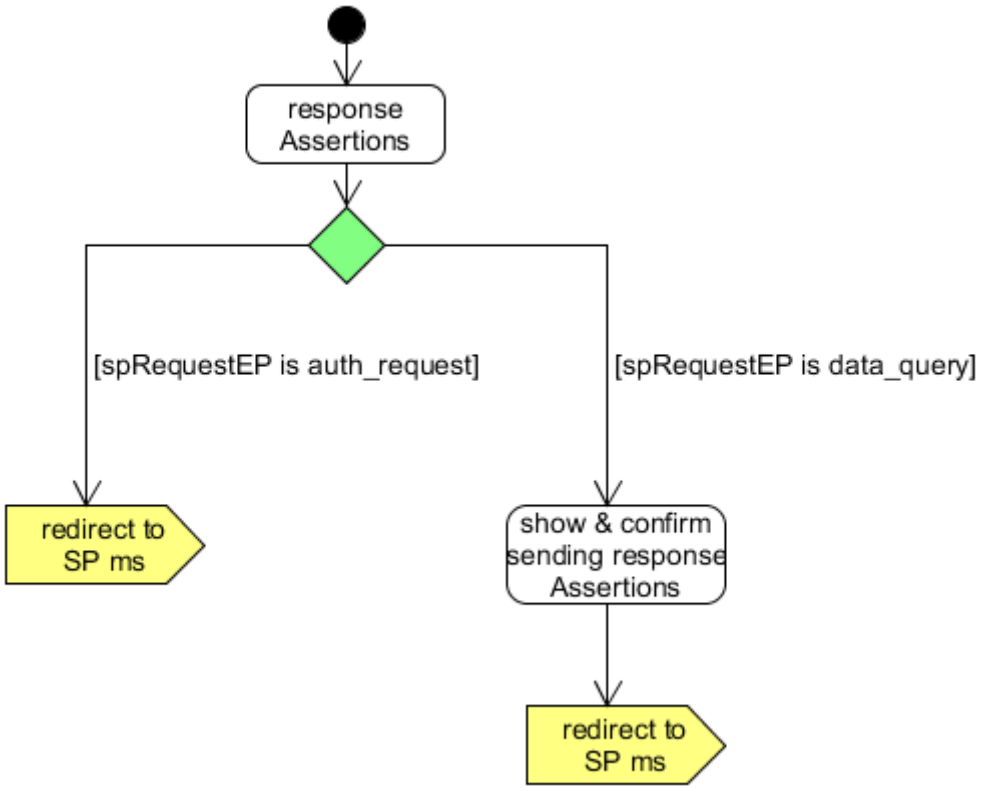


Figure 25 Request Manager response Assertions handling

4.5 Data Models

In this section we define the abstract data models to be used in the SEAL service. The module implementation-specific data models will be specified after the design phase.

4.5.1 Data Set

Generic representation of a set of personal attributes or documents linked to an identity of a subject, retrieved from a source.

- **Data set unique identifier**
- **Subject Id attribute**: name of the attribute that is the ID of the user, a kind of pointer to the attribute ID.
- **Issuer Id attribute**: name of the attribute that is the ID of the entity tht issued the data set, a kind of pointer to the property ID.
- **LoA**: Level of assurance of the authenticity of the data/authentication
- **Issue date**: date when the data set was retrieved from its source

- **Expiration date**: date when the data set becomes invalid
- **Attributes**: [array] of pairs (key, object) representing data belonging to the subject
- **Properties**: [array] of pairs (key, object) representing additional data (metadata) related to the subject or its data attributes

4.5.2 Identity Link

Must uniquely identify two user identities (including their domain, to avoid collisions), and the level of assurance SEAL can provide on them belonging to a same individual. Link is supposed to persist in time (even if the identifier ceases to exist, it was linked to others in the past and must be kept until the user chooses to dismiss it).

- **Unique identifier** of the data set
- **Subject A ID:** a unique identifier in domain A
- **Subject A Issuer**: identifier of the entity that issued subject A (the collision domain of the ID)
- **Subject B ID**: a unique identifier in domain B
- **Subject B Issuer**: identifier of the entity that issued subject B (the collision domain of the ID)
- **Issue date**: date when the link was certified (the date this data set was issued)
- **Link LoA**: level of certainty that both subjects are the same person.

5 Technical Solution

5.1 SEAL Deployment Reference Architecture

The reference deployment for SEAL is depicted in Figure 26 below.

Every microservice is provided in its own container image and deployed on a docker host or on multiple docker hosts. The reference deployment used in development is deployed on one Virtual Machine using docker.

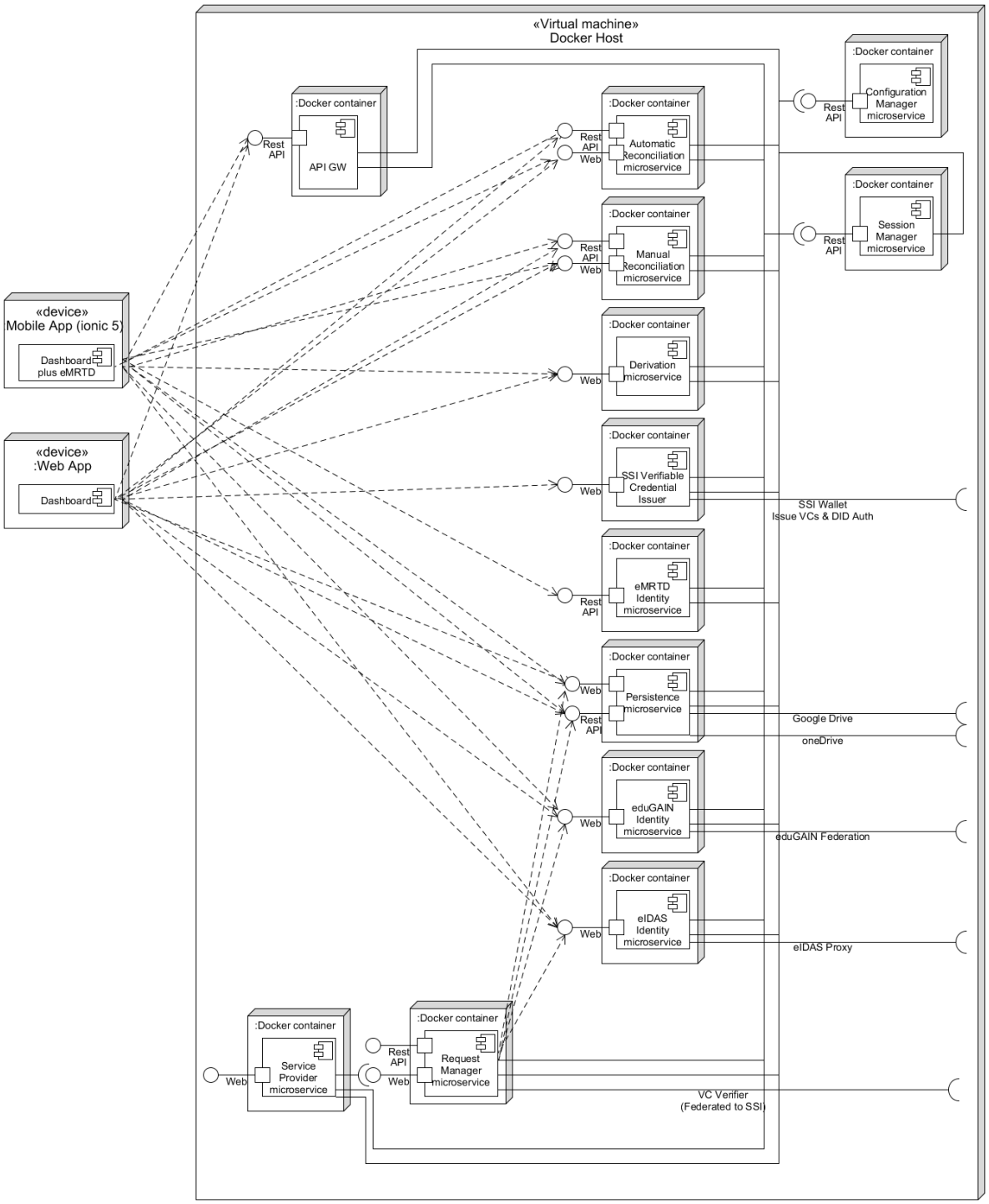


Figure 26 SEAL Deployment Reference Architecture

It is seen that nodes support both web and/or Rest API interfaces. The web interface supports http end point with or without UI and handles front-channel redirects. The Rest API interface handles back-channel calls.

The API GW is the first contact point for the client Dashboard App and performs required business logic and security operations on behalf of the client and responds to the Dashboard App with the access method and end point needed by the client to proceed with the required service.

5.2 Delivery Process

SEAL is an open source project developed on GitHub in this repo: https://github.com/EC-SEAL.

The development guidelines followed in SEAL are detailed in APPENDIX 1 Deployment Guidelines.

There are also guides on how to deploy each specific SEAL container, that can be consulted in the related repository of each microservice in the SEAL GitHub Repo.

The container images developed are downloaded from the docker repository and run with the necessary environment variables, volumes, etc.

For reference, a sample of the docker-compose file used to deploy the SEAL microservices is provided in APPENDIX 2 Docker Compose Example.

5.3 Microservice Design

5.3.1 Modular design

Each functional module in SEAL is implemented in the form of a microservice, which implies:

- It must be deployed as a standalone application.
- Implemented interfaces must follow an HTTP service API pattern.
- Implemented interfaces must follow the security model described in section 5.7.

More than one microservice can be used to implement a functional module, or a single microservice can implement multiple functional modules, and thus, interfaces.

5.3.2 Microservice Communication

In this section, we describe how the microservices communicate among themselves (both front channel and back channel). The model follows the design used in the ESMO project [2].

In both cases, the security framework ensures that:

1. The requestor identity is known and validated by the called microservice.
2. The requestor is authorised to connect to the called microservice.
3. The request information is not tampered with i.e. needs a proof that the request is as the requestor issued it.
4. The request is handed to the expected recipient and not an impersonator.

5.3.2.1 Back Channel Communication

Direct HTTP connections between the requesting microservice and the destination microservice. The first one will open a socket towards the second and they will exchange data in the HTTP protocol. User agent does not intervene here at all.

- Secure HTTP connections only (using trusted certificates)
- HTTP Signature protocol implementation in the client and the server, to validate the identity of both.

5.3.2.2 Front Channel Communication

HTTP Redirections between the requesting microservice and the called microservice. The user agent connects to the requesting microservice using HTTP and then it is instructed to connect to another location, the called microservice. The user agent will then establish an HTTP channel with said microservice.

In front-channel communications, the entity actually calling the destination microservice is the user agent (redirection over the user agent), and not the requesting microservice. As the communication between microservices is indirect, we need different security measures to ensure the origin of the request and its integrity, as in this case not only an external attacker might tamper it, but the same user agent could. To this end:

- The requesting microservice will issue a security token (signed) that will be delivered to the destination microservice via the user agent (the issuer will redirect it via the user agent to a destination ms).
- The exchanged token will be a JSON Web Token (JWT), signed using the JSON Web Signature (JWS) specification.
- The payload of the token will be a JSON structure including the following claims:
    - **tokenID**: uniquely random ID for the request token.
    - **issuer**: the unique ID of the requesting microservice.
    - **destination**: the unique ID of the destination microservice.
    - **issueDate**: date and time of issuing.
    - **notBefore**: time before which the request should not be accepted.
    - **notAfter**: time after which the request should not be accepted.
    - After them, any call specific data can be included.

The above claims are included to provide for basic security checks as described in the following section.

5.4 SEAL Microservice Modules

5.4.1 Configuration Manager

5.4.1.1 Development

This microservice (CM) is being implemented in Java code using Spring. The related github repository can be found at https://github.com/EC-SEAL/conf-manager, where the code updates are done in the default branch (*development*).

In such repository we can find:

The **README** file explains the general structure of this microservice, derived from **ESMO** Project (https://github.com/ec-esmo/ConfMngr). A README_2 file is being used during the development

(https://github.com/EC-SEAL/apigw-client/blob/development/README_2.md), and it is changing accordingly. The following environment variables are used currently:

- ASYNC_SIGNATURE is a Boolean value, if true denotes RSA signing for JWTs, else HS256 signing is conducted.
- KEYSTORE_PATH is the path to the keystore holding the RSA certificate used for signing JWTs.
- KEY_PASS is the password for the certificate.
- STORE_PASS is the password for the keystore containing the certificate.
- HTTPSIG_CERT_ALIAS is the alias of the certificate used for the httpSig protocol.
- SIGNING_SECRET is an HS256 secret used for symmetric signing of JWTs.
- Some information related to the ssl certificate has to be added:
    - SSL_KEYSTORE_PATH
    - SSL_STORE_PASS
    - SSL_KEY_PASS
    - SSL_CERT_ALIAS

Note: there are some traces inherited from ESMO project that will be removed in a next version of the code. It is the case of the dependency on an EWP (Erasmus Without Paper) library, unnecessary in SEAL. See (https://github.com/EC-SEAL/conf-manager/blob/development/scripts/commands.sh).

The **ConfigManager.yaml** file contains the OpenAPI (swagger) specification for this microservice. Note that this file is *a part of* the SEAL_interfaces.yaml (https://github.com/EC-SEAL/interface-specs) which defines all the interfaces of SEAL.

The **Dockerfile** file is intended for packing this microservice as container, which image will be stored in docker-hub under different tags along the development and testing (https://hub.docker.com/repository/docker/mvjatos/seal-cm).

Following the CI/CD directives agreed in the project, a **.travis.yml** has been specified in order to automate the container management and its versions. Apart from the **source directory** (*src*), where is found the code, a **resources directory** is used to store there the configuration files that specifies the SEAL ecosystem. The microservices configuration json file are allocated at /resources. Find msMetadataList.json as an example at https://github.com/EC-SEAL/conf-manager/blob/development/src/test/resources/msMetadataList.json.

The json files describing the different attribute schemas, are put at **/resources/attributeLists**. Examples provided with the deployment are: eduOrg.json, eduPerson.json, eIDAS.json, eMRTD.json, schac.json. See https://github.com/EC-SEAL/conf-manager/tree/development/src/test/resources/attributeLists

The json files with the external entities are found at **/resources/externalEntities**. The available entities until the moment are specified in these files: ACCESSmetadata.json, ATTRSOURCEmetadata.json, AUTHSOURCEmetadata.json, DATAQUERYSOURCESmetadata.json, DERIVATIONmetadata.json, EDUGAINmetadata.json, EIDASmetadata.json,

LINKINGmetadata.json, PERSISTENCEmetadata.json, SSImetadata.json and VCDEFINITIONSmetadata.json. See examples at https://github.com/EC-SEAL/conf-manager/tree/development/src/test/resources/externalEntities

(The json file with the internal configuration inherited from ESMO project can be found at /resources/internal. A LGW_config.json is deployed. This will be removed in next versions as said above.)

Finally, a key store is provided in the **/resources/testKeys** directory.

In summary, a resources directory has to be defined as a volume within the container, with the same structure that has been detailed in the previous paragraphs.

5.4.1.2 Deployment

When the CM is to be deployed, several aspects to take into account are:

- The last version of the container image.
- The necessary environment variables to be set.
- To define a volume where to keep the different configuration files and some ssl certificate information.
- The internal port of this microservice is 8083.

After starting successfully, the related log shows these last lines:

```
2020-05-11 10:47:42.998  INFO 1 --- [           main] eu.seal.cm.ConfMngr2SpringBoot           : Started ConfMngr2SpringBoot in 10.576 seconds (JVM running for 11.356)
2020-05-11 10:47:59.639  INFO 1 --- [nio-8083-exec-5] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring FrameworkServlet 'dispatcherServlet'
2020-05-11 10:47:59.640  INFO 1 --- [nio-8083-exec-5] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2020-05-11 10:47:59.801  INFO 1 --- [nio-8083-exec-5] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 159 ms
```

An example of how to deploy the CM using Docker is given in the following figure:

```
ConfManager:
    image: mvjatos/seal-cm:0.0.3
    environment:
        - KEYSTORE_PATH=/resources/testKeys/keystore.jks
        - KEY_PASS=xxxxx
        - STORE_PASS=xxxxx
        - HTTPSIG_CERT_ALIAS=xxxxx
        - SIGNING_SECRET=xxxxx
        - ASYNC_SIGNATURE=true
        - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
        - SSL_STORE_PASS=xxxxx
        - SSL_KEY_PASS=xxxxx
        - SSL_CERT_ALIAS=xxxxx
    volumes:
        - /SEAL/CM/resources:/resources
    ports:
      - 9083:8083
```

5.4.2 Session Manager

This module has two main functionalities, on the one hand it implements the internal memory of the SEAL service by exposing the appropriate endpoints for managing a SEAL session enabling callers to store and retrieve data and on the other hand it handles the generation of microservice to microservice interaction tokens.

Specifically, the Session Manager implements an internal cache, that the other SEAL microservices can call to store or retrieve arbitrary objects grouped together under a common identifier, the SEAL session.

Additionally, in order to secure microservice to microservice communication the Session Manager supports the generation of security tokens that are used to propagate the session between the SEAL microservices. In more details, these security tokens are encoded as Json Web Tokens (JWT) and contain the internal session identifier, the id of the issuing microservice and the id of the receiving microservice. When secure API endpoints of the SEAL microservices are called they require such a token. The respective microservices validate these security tokens by connecting to the Session Manager and only if validation is successfully done do they proceed with their business flow, by contacting the Session Manager again to retrieve the necessary objects stored in the session cache.

Deployment of the SEAL Session Manager can be made using the following yaml file.

```
SessionManager:
     image: endimion13/esmo-session-manager:0.1.8a
     environment:
         - KEYSTORE_PATH= path to keystore containing SM certificate
         - KEY_PASS=password of the session manager key
         - STORE_PASS=password of the keystore
```

```
            - JWT_CERT_ALIAS=alias for they key used to sign the jwts
            - HTTPSIG_CERT_ALIAS=alias for the key used for http signatures
            - ASYNC_SIGNATURE=true
            - EXPIRES=5
            - CONFIG_JSON= path to json containing the configuration manager
metadata
            - CONFIGURATION_MANAGER_URL=uri of the configuration manager SM
            - MEMCACHED_HOST=memcache host
            - MEMCACHED_PORT=memcache port
            - DATABASE_HOST=docker-mysql
            - DATABASE_USER=root
            - DATABASE_PASSWORD=example
            - DATABASE_NAME=sessionMngr
            - DATABASE_PORT=3306
        links:
            - docker-mysql:mysql
        ports:
          - 8090:8090
        depends_on:
          - docker-mysql

  docker-mysql:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_PASSWORD: example
      MYSQL_DATABASE: sessionMngr
    ports:
      - 3307:3306
```

5.4.3 API GW

5.4.3.1 Development

This microservice (APIGWCL) is being implemented in Java code using Spring. The related github repository can be found at https://github.com/EC-SEAL/apigw-client, where the code updates are done in the default branch (*development*).

In such repository we can find:

The **README** file to take into account is the README_2 (https://github.com/EC-SEAL/apigw-client/blob/development/README_2.md), that could change along the development. The following environment variables are used currently:

- ASYNC_SIGNATURE is a Boolean value, if true denotes RSA signing for JWTs, else HS256 signing is conducted.
- KEYSTORE_PATH is the path to the keystore holding the RSA certificate used for signing JWTs.
- KEY_PASS is the password for the certificate.
- STORE_PASS is the password for the keystore containing the certificate.
- HTTPSIG_CERT_ALIAS is the alias of the certificate used for the httpSig protocol.

- SIGNING_SECRET is an HS256 secret used for symmetric signing of JWTs.
- CONFIGURATION_MANAGER_URL shows the location of the Configuration Manager microservice.
- SESSION_MANAGER_URL is the location of the Session Manager microservice.
- Some information related to the ssl certificate has to be added:
    - SSL_KEYSTORE_PATH
    - SSL_STORE_PASS
    - SSL_KEY_PASS
    - SSL_CERT_ALIAS

The **APIGatewayClient.yaml** file contains the OpenAPI (swagger) specification for this microservice. Note that this file is *a part of* the SEAL_interfaces.yaml (https://github.com/EC-SEAL/interface-specs) which defines all the interfaces of SEAL.

The **Dockerfile** file is intended for packing this microservice as container, which image will be stored in docker-hub under different tags along the development and testing (https://hub.docker.com/repository/docker/mvjatos/seal-apigwcl)

Following the CI/CD directives agreed in the project, a **.travis.yml** has been specified in order to automate the container management and its versions.

Apart from the **source directory** (*src*), where is found the code, a **resources directory** is used to store there the keys used for signing the requests to other microservices, as Configuration Manager or Session Manager. That means a resources directory has to be available as a volume within the container, as it explained following.

5.4.3.2 Deployment

When the APIGWCL is to be deployed, several aspects to take into account are:

- The last version of the container image.
- The necessary environment variables to be set.
- Dependencies on the Configuration Manager and Session Manager.
- To define a volume where to keep the key stores.
- The internal port of this microservice is 8053.

After starting successfully, the related log shows these last lines:

```
2020-05-19 10:30:34.160  INFO 1 --- [           main] eu.seal.apigw.cl.ApiGwCl2SpringBoot      : Started ApiGwCl2SpringBoot in 9.528 seconds (JVM running for 10.538)
2020-05-19 10:31:06.253  INFO 1 --- [nio-8053-exec-5] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring FrameworkServlet 'dispatcherServlet'
2020-05-19 10:31:06.254  INFO 1 --- [nio-8053-exec-5] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2020-05-19 10:31:06.275  INFO 1 --- [nio-8053-exec-5] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 21 ms
```

An example of how to deploy the APIGWCL using Docker is given in the following figure:

```
apigwcl:
    image: mvjatos/seal-apigwcl:test
    environment:
        - KEYSTORE_PATH=/resources/testKeys/keystore.jks
        - KEY_PASS=xxxxx
        - STORE_PASS=xxxxx
        - HTTPSIG_CERT_ALIAS=xxxxx
        - SIGNING_SECRET=xxxxx
        - ASYNC_SIGNATURE=true
        - CONFIGURATION_MANAGER_URL=https://vm.project-seal.eu:8083
        - SESSION_MANAGER_URL=http://SessionManager:8090
        - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
        - SSL_STORE_PASS=xxxxx
        - SSL_KEY_PASS=xxxxx
        - SSL_CERT_ALIAS=xxxxx
    volumes:
        - /SEAL/APIGWCL/resources:/resources
    links:
        - ConfManager:vm.project-seal.eu
        - SessionManager:SessionManager
    ports:
      - 9053:8053
    depends_on:
      - SessionManager
      - ConfManager
```

5.4.4 Persistence

The Persistence microservice module is responsible for the encryption and decryption of the user's personal data store, making possible for him to save the SEAL datastore in several different places, namely in the local browser storage, the local mobile storage, Google Drive, One Drive, and others that we are expanding, allowing him to use them when needed without anyone having access to the information contained in it.

This microservice is being developed in Go Lang and the repository can be found in the Github repository of EC-SEAL project (https://github.com/EC-SEAL/perseal).

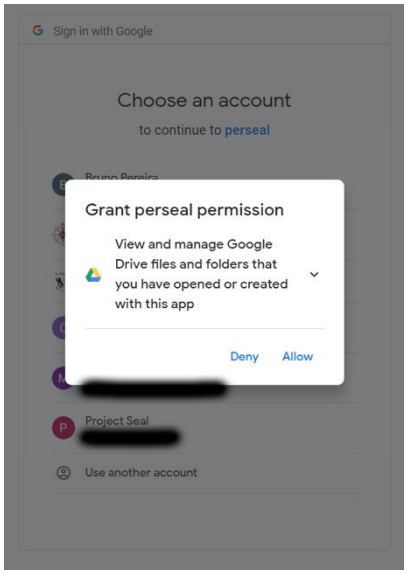
The repository has a README.md (https://github.com/EC-SEAL/perseal/blob/master/README.md) file which has a small description on how to use build and run the microservice and which endpoint should be used in each use case that the microservice is needed, along with the parameters required and the returned structures. This way, the other microservices can easily develop their interactions with this one.

In the env-example file you can find the environmental variables we use so there are no hardcoded credentials and keys:

- AUTH_URL
- REDIRECT_URL
- FETCH_TOKEN_URL
- CREATE_FOLDER_URL
- CREATE_FOLDER_URL

- GET_FOLDER_URL
- SM_ENDPOINT
- GOOGLE_DRIVE_CLIENT
- ONE_DRIVE_CLIENT
- ONE_DRIVE_SCOPES
- GOOGLE_DRIVE
- ONE_DRIVE
- PERSEAL_INT_PORT
- PERSEAL_EXT_PORT
- PERSEAL_EMAIL

The encryption and decryption of the file is made by using the current, up-to-date libraries, using the latest technology and standards, providing a secure implementation. The encryption and decryption are made with the AES algorithm with CBC block, using the hash of the clear text password with SHA256 algorithm. Also, the connections to the Google Drive and One Drive environments are made according to their policy, using OAuth 2.0 and asking for SEAL app permission to access the files.



The microservice is split in two different parts, the login which is developed in Go Lang and implements all the endpoints and functions and second part which is the UI and is connected to the first one to provide a small interface for user operations, like the introduction of the password used to encrypt or decrypt the datastore.

The UI was developed in AngularJS, for the moment, and provides a simple interface for the user, for small operations, as you can see in the image below.

5.4.5 Derivation

5.4.5.1 Development

This Derivation-id UUID microservice is being implemented in Java using Spring Framework. The related github repository can be found at https://github.com/EC-SEAL/derivation-id, where the code updates are being included on a side branch, as per the use case sequence diagram UC601.

In the repository we can find:

The README (link): Providing the steps to run and test the service.

As it is included in the readme, the following environment variables are used currently:

- ASYNC_SIGNATURE is a Boolean value, if true denotes RSA signing for JWTs, else HS256 signing is conducted.
- KEYSTORE_PATH is the path to the keystore holding the RSA certificate used for signing JWTs.
- KEY_PASS is the password for the certificate.
- STORE_PASS is the password for the keystore containing the certificate.
- HTTPSIG_CERT_ALIAS is the alias of the certificate used for the httpSig protocol.
- SIGNING_SECRET is an HS256 secret used for symmetric signing of JWTs.
- CONFIGURATION_MANAGER_URL shows the location of the Configuration Manager microservice.
- SESSION_MANAGER_URL is the location of the Session Manager microservice.

Derivation-id.yaml (link) file contains the OpenAPI (swagger) specification for this microservice. Note that this file is *a part of* the SEAL_interfaces.yaml (https://github.com/EC-SEAL/interface-specs) which defines all the interfaces of SEAL.

The **Dockerfile (link)** file is intended for packing this microservice as container, which image will be stored in docker-hub under different tags along the development and testing (https://hub.docker.com/repository/docker/cbuendiaatos/derivation-id)

5.4.5.2 Deployment

To run the Derivation ID microservice, two different options are provided. In either of them, it is strictly necessary to have the previously mentioned environment variables set. To facilitate this task, an example environment file has been provided here: .

Having this requirement, the service can run in development by using maven:

- **Via Maven**: Two different commands are opened to run the service directly from maven:
  - Running tests: $ mvn test -B
  - Running the service: $mvn spring-boot:run -X

As this service is still in development, integration into the virtual machine deployment docker-compose information will be done in future steps of development. This information will be completed in an updated version of the deliverable.

5.4.6 Identity Linking / Reconciliation

Reconciliation modules receive a request issued by the own user, including two datasets he wants to be considered for reconciliation.

The modules have a register/process/retrieve kind of interface, to allow supporting time-detached processing of requests and the connection of thrusted third parties. It also has an interface to allow file transfer and text-communication between the requester and the module, to support interaction with remote human agents from the same SEAL clients.

The manual module implements a complete interface to support the login, request selection and processing of linking requests, to support the action of human officials of the deployer organisation.

The automated module implements a data set processing engine, based on configurable rules, that allows multiple transformations and semantic pairings between the attributes in the datasets, to maximise the matching possibilities. The rules are defined as generic rules for a given pair of data sources. The module implements a transliteration engine as well, to support comparisons between strings encoded in any of the EU official alphabets (Latin, Greek and Cyrillic). Finally, the module implements an approximate string comparison algorithm based on the Levenshtein-Damerau edit distance. The module allows for the connection of additional implementations, and in the future will implement a mixed approach of edit distance with unit-comparison algorithm, to make up for the blind spot caused by strings with switched words on it (like names and surnames)

You can find the deployment information and module documentation in the module repository Readme files.

For the automated linking module:

https://github.com/EC-SEAL/reconciliation

For the manual linking application:

https://github.com/EC-SEAL/link-service

5.4.7 SP request handling

SP Module implements multiple interfaces to validate and process secure authentication or data query requests from Service Providers to SEAL. The implemented interfaces are: OIDC, standard SAML2 and a SAML2 with eIDAS extension interface.

The deployment information and module documentation can be found in the module repository Readme files: https://github.com/faragom/ESMO_SAML/blob/master/README.md.

5.4.8 VC Issuer

This module acts as the enabler of Self Sovereign Identity (SSI) for the SEAL Service. Specifically, this module implements a uPort[1] compatible Agent exposing the required APIs and web views that are needed to issue W3C Verifiable Credentials to a user's SSI wallet. The attributes contained within these credentials are retrieved from the SEAL Service internal session. If no such data is found in session, the VC Issuer contains the appropriate user interfaces to guide their retrieval by integrating with the corresponding SEAL Services microservices. Also, this module exposes APIs for handling DID authentication and storing the responses in the SEAL service session, separately from the VC issuance flow. This functionality ensures that DID authentication can take place at the start of a SEAL Service session, ensuring the security of the SSI issuance.

The handling of the required cryptographic material for the signing of the VC's is abstracted in such a way that it enables the integration with a Qualified eIDAS Trust Service Provider (QTSP) supporting remote eIDAS QSeals. This way, the SEAL service is capable of anchoring trust to the eIDAS network for the Verifiable Credentials issued by the service.

Finally, this module manages the revocation of credentials (if such needs arise) by integrating with a permissioned blockchain and securely storing revocations to a smart contract implementing the systems Revocation Registry.

The SEAL VC Issuer module is implemented in Node.js and Next.js (specifically Node is used as the backend server exposing all the required endpoints and also serves the Next.js app). This stack enables for the user interfaces to be optimized as a single page application (SPA) with server side rendering (SSR). Additionally, using this technology stack the actual user attributes and the generation and issuance of the VCs is handled on the back end of the system, relying only on backend data sources. In this way the securing of the issuance flows is ensured. Finally, the SEAL VC Issuer module is Dockerized for easier deployment. Specifically, the source code and the dockerhub image are available in the following links:

- Github repository: https://github.com/EC-SEAL/vc-issuer
- Dockerhub image: https://hub.docker.com/r/endimion13/seal-issuer

[1] uPort consists of an open source W3C Verifiable Credential compatible SSI solution, providing users with free to use SSI wallets available in all app stores (googleplay and appstore). For more details please refer to: https://www.uport.me/

Deployment of the SEAL issuer can be made using the following yaml file.

```
uportissuer:
  image: endimion13/seal-issuer:0.0.3d
  environment:
    NODE_ENV: "production"
    ENDPOINT:  the server running the service, e.g. https://dss1.aegean.gr
    HTTPS_COOKIES: "true"
    BASE_PATH: when deployed behind a reverse proxy, add the base url here
    SENDER_ID: SEAL service senderId, i.e. the id of this microservice
    MEMCACHED_URL: the memcached instance url used to maintain session on the
backend
    SEAL_CONF_URI : Configuration manager uri
    SEAL_CONF_PORT: optional Configuration manager port, defaults to '9090'
  ports:
    - 4000:3000

memcached:
  image: sameersbn/memcached:1.5.6-2
  ports:
    - 11112:11211
```

5.4.9 VC Verifier

The VC Verifier consists of an OpenId Connect (OIDC)/SAML server based on Keycloak[2][3], customized in the context of SEAL to enable user authentication using SEAL-issued verifiable credentials. Specifically, this module consists of two sub-modules: the SSI SDK helper and the SSI keycloak-plugin. These modules are to be deployed in the same host with only the SSI Keycloak-plugin exposing endpoints to the network (communication between them takes place as intar-container with no public network access). In more details:

- The SSI SDK helper is written in Node.js and implements a uPort SSI verifying Agent, capable of generating the necessary VC disclosure requests (i.e. request VCs from a user's wallet) and validate the VC disclosure responses (i.e. validate the VCs received as a response to a disclosure request). This module integrates with the permissioned blockchain deployed in the context of SEAL that contains the Revocation Registry. As an additionally mean of verification this module integrates with the SSI eIDAS Bridge[4] to ensure that the certificate used to sign the received VCs is still valid (by eIDAS).
- The SSI Keycloak-plugin written in Java, consists of a custom Keycloak authenticator capable of translating the requested client scopes into SSI VC disclosure requests by connecting with the SSI SDK helper module, and after the VC disclosure response has been parsed by the SSI

[2] Keycloak is an Open Source Identity and Access Management https://www.keycloak.org/

[3] An enhanced security version of Keycloak was developed in the context of the ESMO project (CEF AGREEMENT No INEA/CEF/ICT/A2017/1451951 http://www.esmo-project.eu/) and is being reused for this project

[4] https://joinup.ec.europa.eu/collection/ssi-eidas-bridge/document/ssi-eidas-bridge-api-documentation

SDK helper module, translate the response into either OIDC or SAML claims and finally propagate these to the client

Specifically, this service is deployed at the Athens ESMO GW[5]. The Athens ESMO GW is a customized Keycloak server offering enhanced security and privacy features over a typical such deployment and was developed during the ESMO project[6] and is maintained by the Greek Ministry of Education in collaboration with GUNet[7].

By adopting this architecture, the service providers (i.e. the HEI e-services connecting to SEAL) minimize their barrier to entry to allow their users to authenticate using SEAL-issued VCs. They can continue to use the protocols/stack they are familiar and comfortable with (either OIDC or SAML) thus significantly simplifying the integration process.

Deployment of the SEAL verifier can be made using the following yaml file:

```
memcached:
    image: sameersbn/memcached:1.5.6-2
    ports:
      - 11111:11211

  uporthelper:
    image: endimion13/uport-verifier:0.0.2j
    environment:
      PRODUCTION: "true"
      KEYCLOAK: ${keycloak deployment url}/auth/realms/{keycloak SSI realm}/ssi-
sp/ssiResponse
      KEYCLOAK_MOBILE:${keycloak deployment url}/auth/realms/{keycloak SSI
realm}/ssi-sp/proceedMobile
      MEMECACHED: memcached:11211
    links:
      - memcached
    ports:
      - 3000:3000

 keycloak:
      image:  jboss/keycloak:latest
      environment:
        PROXY_ADDRESS_FORWARDING: 'true'
        # additional keycloak deployment parameters are skipped
        UPORTHELPER: container uri of the uPort skd helper
        CALLBACK_MOBILE:  ${keycloak deployment URI}/auth/realms/{SSI keycloak
REAL}/ssi-sp/proceedMobile
        CALLBACK: ${keycloak deployment URI}/auth/realms/{SSI keycloak REAL}/ssi-
sp/uportResponse
        AUTH_PROCEED: ${keycloak deployment URI}/auth/realms/{SSI keycloak
REAL}/ssi-sp/protocol/openid-connect/auth
        EVENT_SOURCE: ${keycloak deployment URI}/auth/realms/{SSI keycloak
```

[5] https://esmo-gateway.eu/about/
[6] CEF AGREEMENT No INEA/CEF/ICT/A2017/1451951 http://www.esmo-project.eu/
[7] https://www.gunet.gr/en/

```
REAL}/ssi-sp/subscribe
        SSI_REPLY_POST: ${keycloak deployment URI}/auth/realms/{SSI keycloak
REAL}/ssi-sp/proceed
      ports:
        - 8081:8080
      depends_on:
        - mysql
        - memcached
      links:
        - memcached:memcached


memcached:
    image: sameersbn/memcached:1.5.6-2
    ports:
      - 11111:11211

  uporthelper:
    image: endimion13/uport-verifier:0.0.2j
    environment:
      PRODUCTION: "true"
      KEYCLOAK: https://dss1.aegean.gr/auth/realms/SSI/ssi-sp/ssiResponse
      KEYCLOAK_MOBILE: https://dss1.aegean.gr/auth/realms/SSI/ssi-sp/proceedMobile
      MEMECACHED: memcached:11211
    links:
      - memcached
    ports:
      - 3000:3000
```

After the keycloak container is running the SSI Keycloak-plugin needs to be copied to the deployments folder of the container. For example this could be achieved as follows :

```
$ docker cp ssi-plugin.ear
keycloak_keycloak_1:/opt/jboss/keycloak/standalone/deployments
```

5.4.10 Request Manager

5.4.10.1 Development

This microservice (RM) is being implemented in Java code using Spring. The related github repository can be found at https://github.com/EC-SEAL/request-manager, where the code updates are done in the default branch (*development*).

In such repository we can find:

The **README** file to take into account is the README (https://github.com/EC-SEAL/request-manager/blob/development/README.md)), that could change along the development. The following environment variables are used currently:

- ASYNC_SIGNATURE is a Boolean value, if true denotes RSA signing for JWTs, else HS256 signing is conducted.
- KEYSTORE_PATH is the path to the keystore holding the RSA certificate used for signing JWTs.
- KEY_PASS is the password for the certificate.
- STORE_PASS is the password for the keystore containing the certificate.

- HTTPSIG_CERT_ALIAS is the alias of the certificate used for the httpSig protocol.
- SIGNING_SECRET is an HS256 secret used for symmetric signing of JWTs.
- CONFIGURATION_MANAGER_URL shows the location of the Configuration Manager microservice.
- SESSION_MANAGER_URL is the location of the Session Manager microservice.
- Some information related to the ssl certificate has to be added:
    - SSL_KEYSTORE_PATH
    - SSL_STORE_PASS
    - SSL_KEY_PASS
    - SSL_CERT_ALIAS

The OpenAPI (swagger) specification for this microservice is defined in the SEAL_interfaces.yaml (https://github.com/EC-SEAL/interface-specs) file, which defines all the interfaces of SEAL.

The **Dockerfile** file is intended for packing this microservice as container, which image will be stored in docker-hub under different tags along the development and testing (https://hub.docker.com/repository/docker/rccatos/seal-rm)

5.4.10.2 User Interface

The Request Manager module supports a web interface for the user to decide how to respond to the verification request from the SP and is described below.

SP Request

The Request Manager implements an end point user interface for presenting the request from the SP for the user to authenticate against eIDAS or eduGAIN or provide requested identity attributes from all available sources also presented, namely SSI Wallet, User´s Personal Data Store, eduGAIN federation, eIDAS.

The SP Request mobile and web views are supported as shown in the following figures.

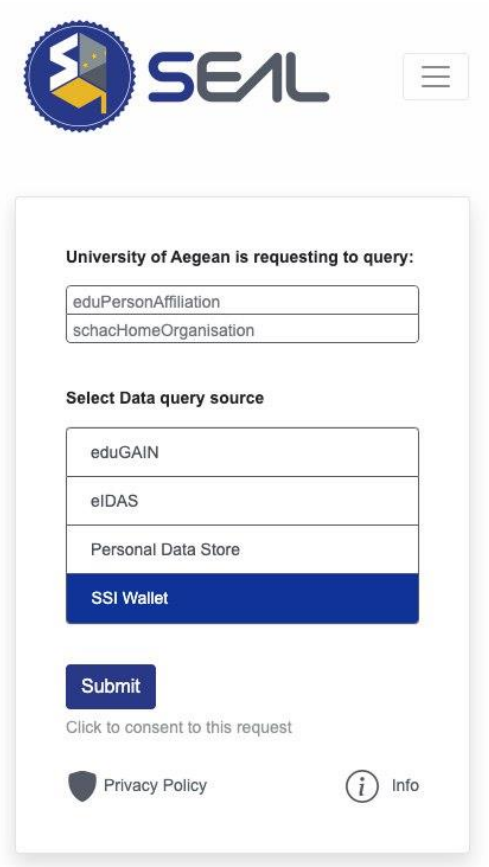


Figure 27 SP Request Mobile User Interface

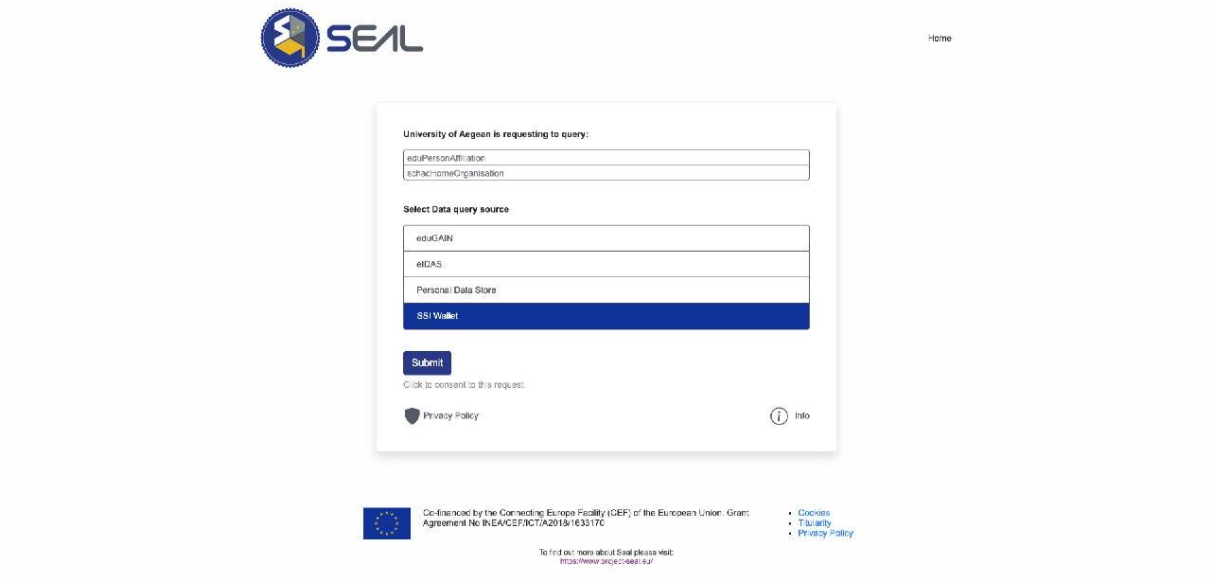


Figure 28 SP Request Desktop User Interface

Response Assertions :

The response received from the attribute source is presented to the user where they can choose to deselect some information and then provide consent to deliver the response back to the Service Provider. Response mobile and web views are supported as shown in the following figures.

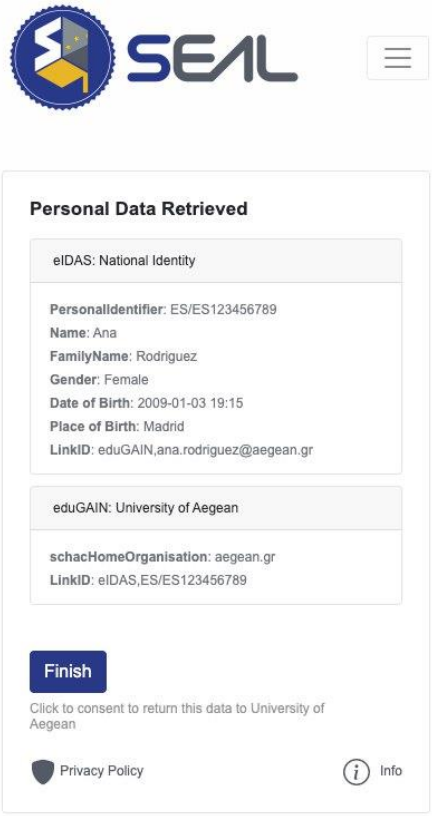


Figure 29 Mobile UI Response Assertions

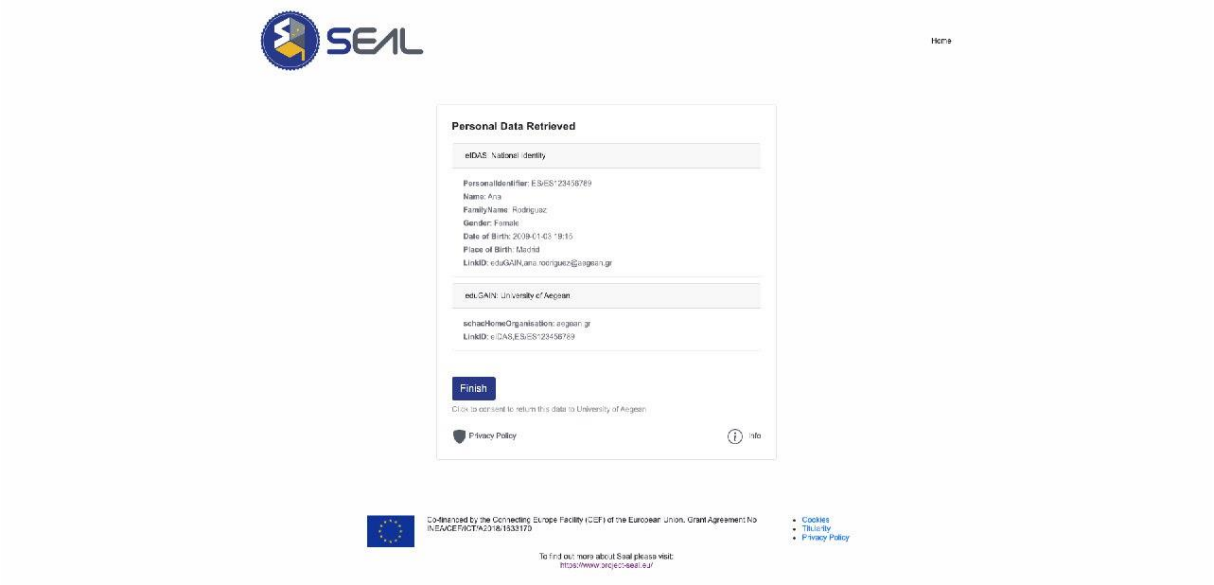


Figure 30 Desktop UI Response Assertions

5.4.10.3 Deployment

When the RM is to be deployed, several aspects to take into account are:

- The last version of the container image.
- The necessary environment variables to be set.
- Dependencies on the Configuration Manager and Session Manager.
- To define a volume where to keep the key stores.
- The internal port of this microservice is 8063.

5.5 SSI / Blockchain Implementation

For the scope of the project a blockchain infrastructure is needed to contain the metadata of the SEAL Service VC Issuer module and most importantly to maintain the Revocation Registry (i.e. a list of credential identifiers that have been revoked by the SEAL VC Issuer module). These two features enhance the security and applicability of the SEAL Service by supporting fine grained control over the lifecycle of the issued VCs to an extent that is not possible by simply using PKI. For example, consider the case where a student is issued a VC affiliating them with an HEI until the end of the year. However, the student mid semester decides to quit. Using a traditional PKI approach revocation of the VC would not be possible without invalidating all issued credentials. However, by maintaining a tamper proof Revocation Registry on the SEAL Service connected blockchain the micromanagement of issued credentials is possible.

Specifically, in the context of SEAL a permissioned Ethereum blockchain is being deployed by the consortium. The reason behind this decision lies on the fact that once the European Blockchain Services

Infrastructure (EBSI)[8] becomes available the SEAL blockchain implementation could be moved to the EBSI infrastructure to ensure the maintainability of the project and also benefit from higher levels of security than is possible to achieve within the context of SEAL.

In more details, three nodes are being set up (at the universities of Aegean, Malaga and Porto). The blockchain implementation being used is Hyperledger Besu. Hyperledger Besu is an Ethereum client designed to be enterprise-friendly for both public and private permissioned network use cases. This implementation supports several consensus algorithms including Proof of Work, and Poor of Authority (specifically through the IBFT, IBFT 2.0, Etherhash, and Clique algorithms). In SEAL the Clique (Proof of Authority) consensus protocol was chosen (due to the number of available nodes). To strengthen the security of the system by countering against collusion and to add extra assurances to the historical finality of confirmed transactions Tethering is being considered[9]. Tethering consists of the process of appending the hashes of the private blockchain state (at predefined time intervals) to the public Ethereum network.

5.6 Internal Interfaces Definition

The formal specification of all standard module interfaces in SEAL Service is written in the OpenAPI specification [4] and is published in the SEAL GitHub Repo here: https://github.com/EC-SEAL/interface-specs/blob/master/SEAL_Interfaces.yaml.

The rest of this section gives an overview of the APIs supported by SEAL for each interface.

[8] https://joinup.ec.europa.eu/collection/blockchain-egov-services/about
[9] https://docs.kaleido.io/kaleido-services/tether/

5.6.1 Session Manager Interface

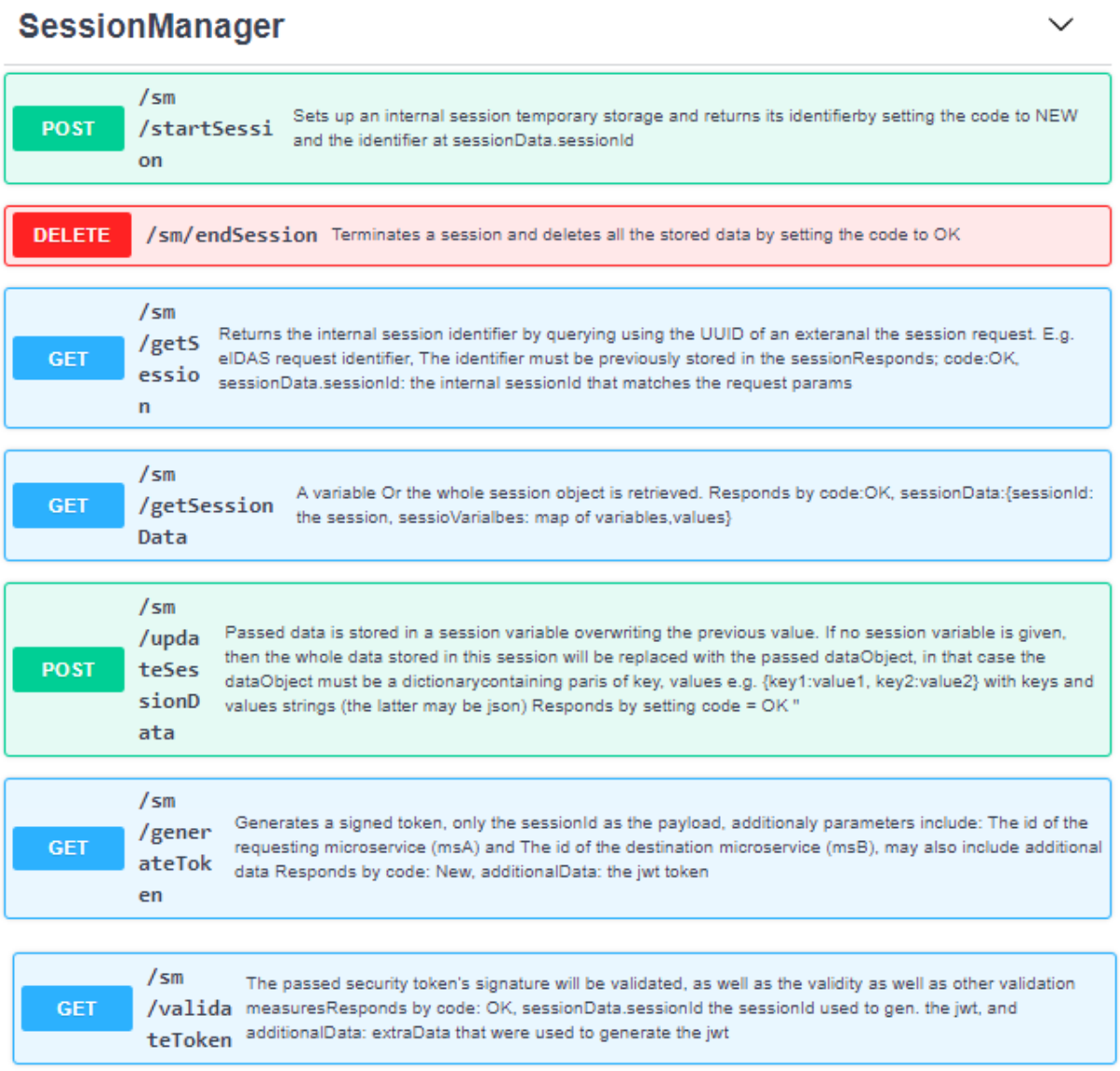


Figure 31 Session Manager APIs

5.6.2 Configuration Manager Interface

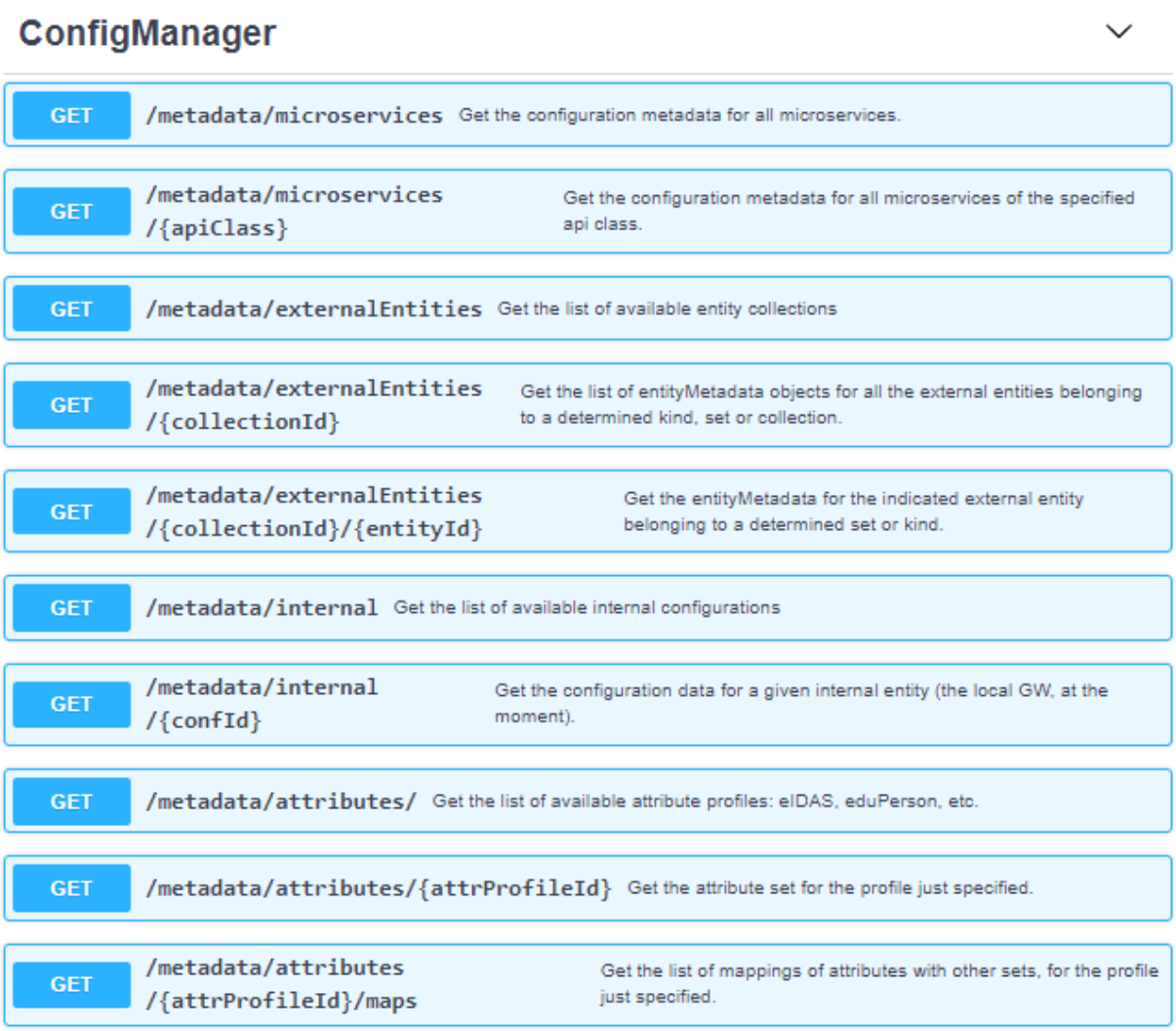
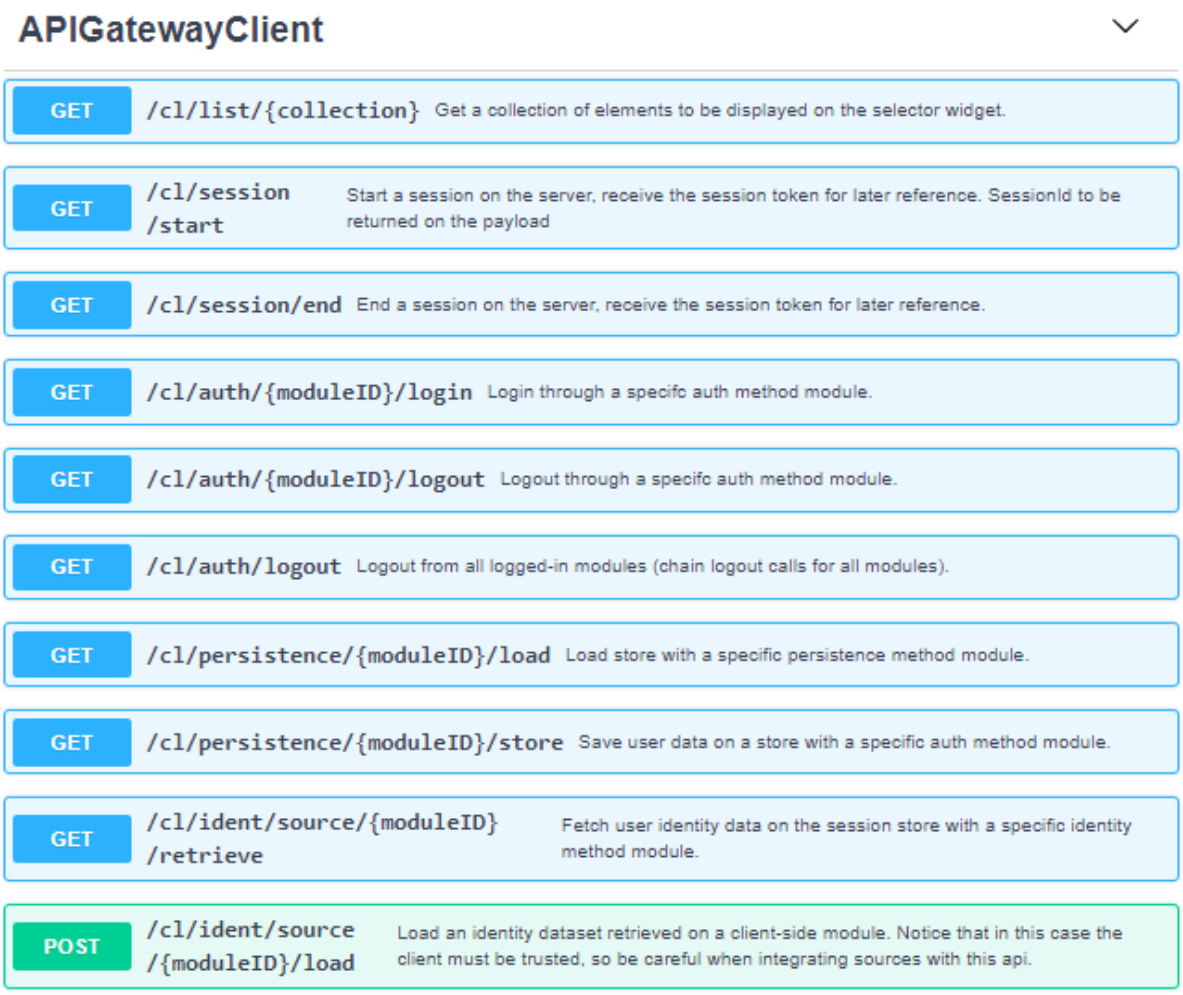


Figure 32 Configuration Manager APIs

5.6.3 APIGW Interface



GET	/cl/ident/derivation/{moduleID}/generate	Generate a derived identity through a specific method module.
GET	/cl/ident/mgr/list	Get the list of user identity data sets currently loaded or fetched in session.
GET	/cl/ident/mgr/{datasetID}/refresh	Update a retrieved or derived identity already on the session store through the specific method module used to obtain it.
GET	/cl/ident/mgr/{datasetID}/delete	Delete a retrieved or derived identity already on the session store through the specific method module used to obtain it.
GET	/cl/ident/mgr/{datasetID}/revoke	Revoke a retrieved or derived identity already on the session store through the specific method module used to obtain it.
POST	/cl/ident/linking/{moduleID}/request	Request two data sets to be reconciled through a specific method module.
GET	/cl/ident/linking/{moduleID}/{requestId}/status	Request the status of a reconciliation request to a specific method module.
GET	/cl/ident/linking/{moduleID}/{requestId}/cancel	Request to cancel a reconciliation request on a specific method module.
GET	/cl/ident/linking/{moduleID}/{requestId}/result	Fetch the result of a reconciliation request to a specific method module.
POST	/cl/ident/linking/{moduleID}/{requestId}/files/upload	Upload a supporting file to a specific method module.

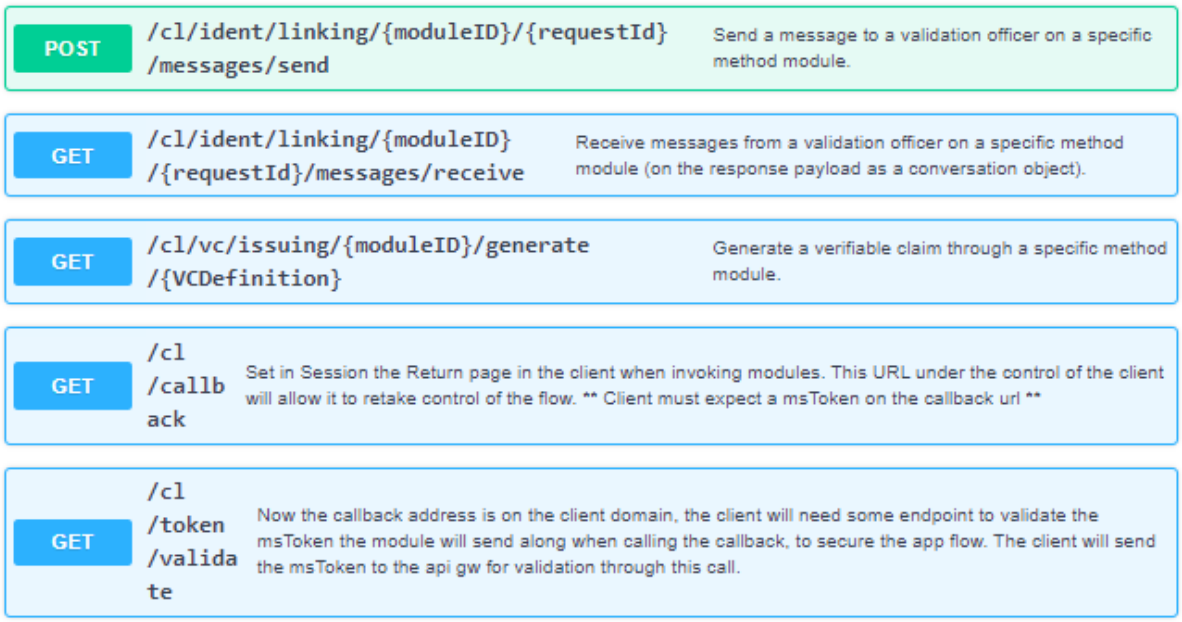


Figure 33 APIGW APIs

5.6.4 Identity Interface

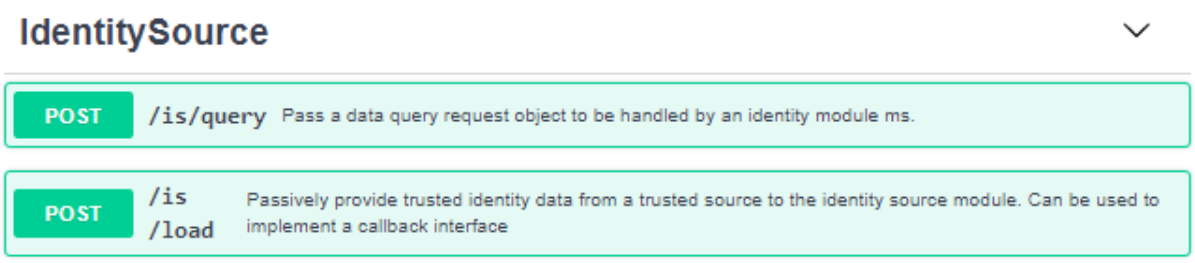


Figure 34 Identity Source APIs

5.6.5 Authentication Source Interface

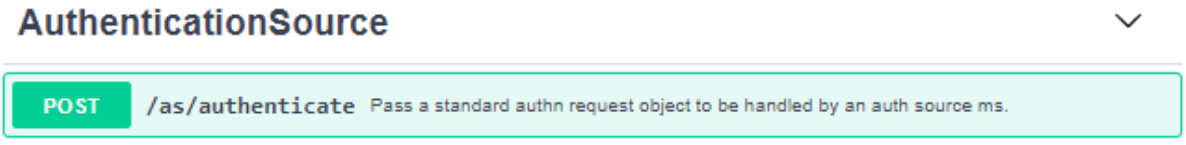


Figure 35 Authentication Source API

5.6.6 Persistence Interface

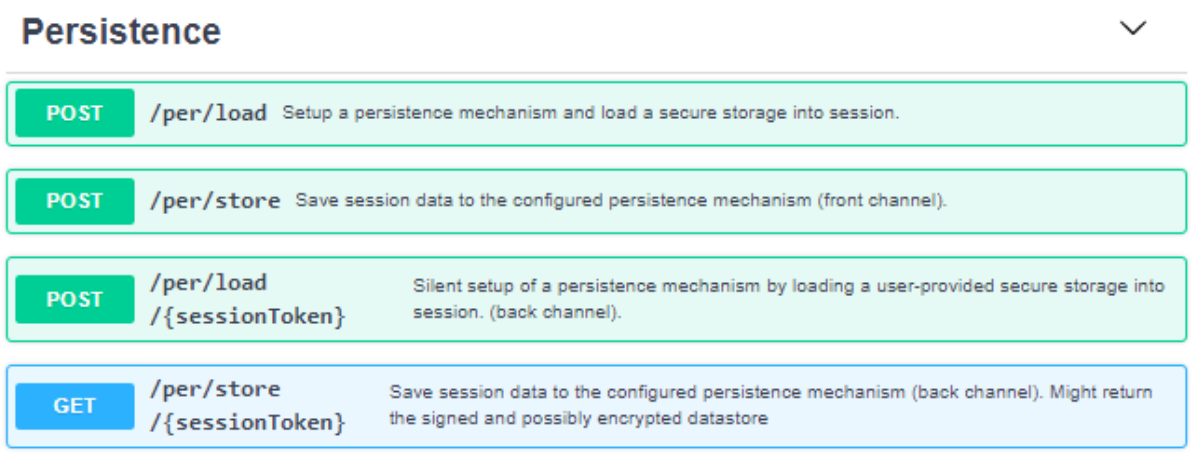


Figure 36 Persistence APIs

5.6.7 Verifiable Credential Issuing Interface

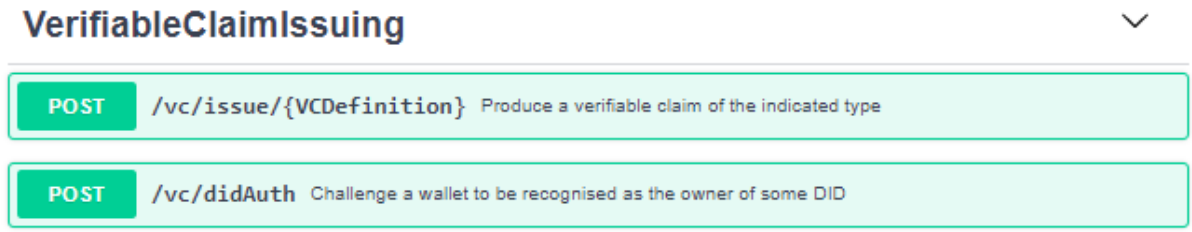


Figure 37 Verifiable Credential Issuing APIs

5.6.8 Derivation Inteface

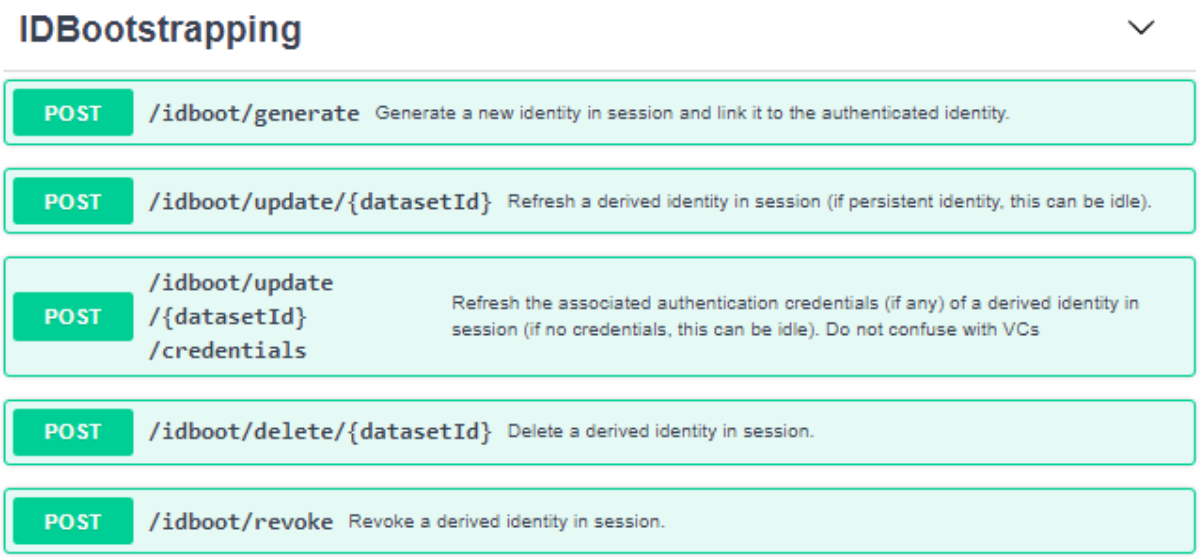


Figure 38 Derivation APIs

5.6.9 Request Manager Interface

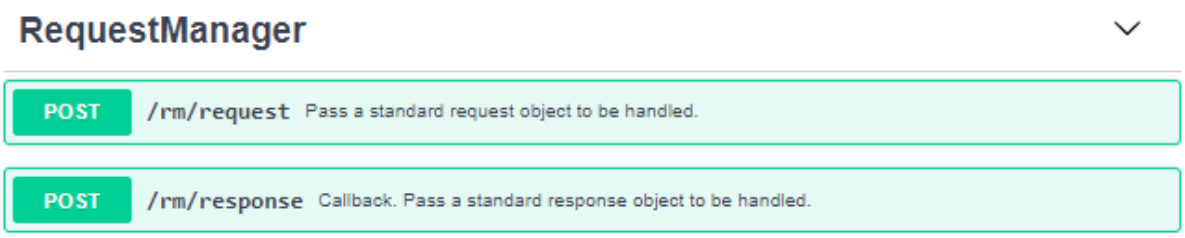


Figure 39 RM APIs

5.6.10 SP Verification Response Interface

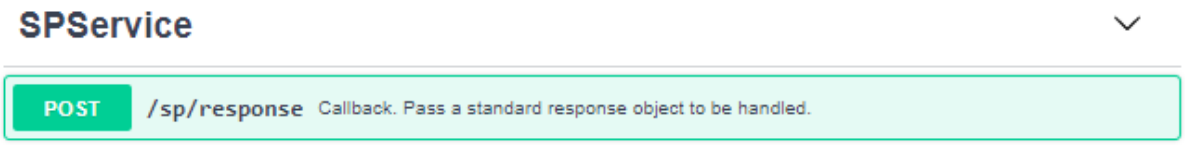


Figure 40 Service Provider Verification Interface7

5.6.11 Linking Interface

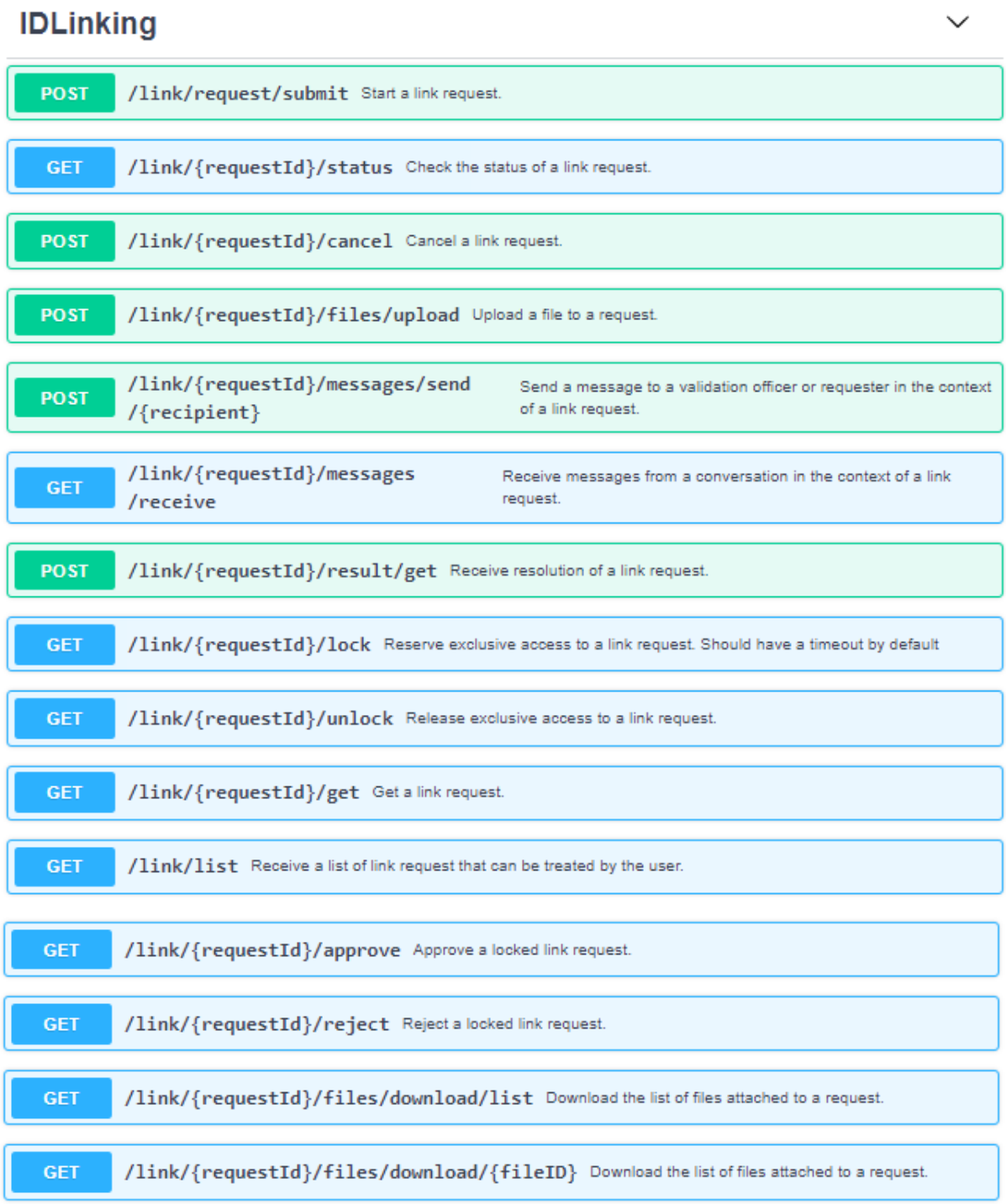


Figure 41 ID Linking APIs

5.6.1 Linking Client Interface

APIGatewayLink

GET	/linkapp/list/{collection}	Get a collection of elements to be displayed on the selector widget.
GET	/linkapp/session/start	Start a session on the server, receive the session token for later reference.
GET	/linkapp/session/end	End a session on the server, receive the session token for later reference.
GET	/linkapp/auth/{moduleID}/login	Login through a specifc auth method module.
GET	/linkapp/auth/logout	Logout on the used auth method module.
GET	/linkapp/modules/list	Get a list of pending link requests that can be handled, from all modules by the authenticated official.
GET	/linkapp/requests/{moduleID}/list	Get a list of pending link requests that can be handled by the authenticated official on a specific module.
GET	/linkapp/requests/{moduleID}/{requestId}/lock	Lock a pending link request to prevent other officials from handling it, on a specific module.
GET	/linkapp/requests/{moduleID}/{requestId}/unlock	Release a locked a pending link request so other officials can handle it, on a specific module.
GET	/linkapp/requests/{moduleID}/{requestId}/get	Get a pending link request from a specific module.

Figure 42 Linking Client APIs

5.7 Security Implementation

In this section, we describe all the security aspects of data storage and communications, both internal (between microservices) and external (between entities). The model follows the design used in the ESMO project [2].

5.7.1 Data Storage Security

SEAL does need user persistent data for the operation of its service; however it provides the user with the means to securely store their data on their local devices or cloud service, or indeed to use a 3rd party SSI wallet.

Data needs in SEAL can be classified as follows:

- Public data, not involving plain personal information.
- Personal data managed only by the user.
- Personal data managed by a third party.

Guiding principles are as follows:

- Minimise the stored data.
- Write a proof of each transaction for later traceability.
- To ensure data integrity, perform periodic checks with the proofs over the data.

Most of the data, and the most critical from the security point of view, is personal data managed by the user only, so it will be stored in a distributed storage layer, based on the device of the user (browser storage, local file, mobile storage, cloud storage account).

The public data will be written on distributed ledgers or on central service storage, although all centrally stored data should have a proof written on a ledger, to facilitate integrity checks.

The personal data to be managed by a third party must be stored on the central service storage, and should:

- Be written only if the user consents to.
- Minimise the time it resides there.
- Write a traceable proof of all the transactions it carries out.
- Once the third party has finished with the data, it must be returned to the user distributed storage.

5.7.2 Internal Communication Security

The lightweight microservice model developed in ESMO, allows to centralise the security handling on a single microservice, reducing development overhead for the rest of micro-services. For convenience, we use the Session Manager, the one that keeps session state. All the calls to this microservices are in backchannel, and only other microservices will be authorised to call this microservice; no external entity or the user agent will have access to it.

This microservice will be the only microservice able to issue and validate a security token for the front-channel connections, and it is a JWT security token.

So, any sensitive information the microservices exchanged, will be done through the session held at the Session Manager. The exchanged token payload, will be just the session ID. The caller will have to know which variables to write on the interface call, and the interface implementer, which variables to read.

When a microservice “A” wants to communicate with a microservice “B”:

- “A” will require to start a session with the Session Manager (if it is the first microservice in the logic). A session identifier “S” will be generated by Session Manager and delivered to “A”.
- “A” will store any sensitive information addressed to “B” in the “S” session.
- “A” will request a JWS token to Session Manager, indicating the destination microservice “B” and the payload content (any relevant call parameter not stored in session and the session token “S”). Session Manager will return the signed token “T”.
- “A” will redirect the user agent to “B” carrying the token “T”.
- “B” will receive “T” and submit it for validation to Session Manager.
- Session Manager will validate the signature and then all the claims to ensure the application logic is valid:
    - Current date is between the validity dates.
    - Token ID has not been presented before (to avoid replay attacks)
    - “B” was the microservice that submitted for validation (and also the legitimate recipient of the token)

- "A" is an authorised caller to "B" (Session Manager will keep the access lists for all the microservices).

- If some check has failed, Session Manager will return the error to "B", and "B" will terminate the processing, as the received token was not valid or legitimate.
- If all checks passed, Session Manager will return the effective payload to "B" (including the session identifier "S"), so it can be used to go on with the logic.
- "B" will use "S" to retrieve any needed session stored data.

5.7.3 External Communication Security

In SEAL, all modules responsible for external communications, follow these guidelines:

- All personal information requestors must be authenticated and authorised. For example, authenticated through signed challenges/requests, and authorised because their public key has been added to a trust list through a trusted method.
- All sources of data must be authenticated and authorised. For example, authenticated through signed challenges/requests, and authorised because their public key has been added to a trust list through a trusted method.
- All data stores must be authenticated, for example, through an SSL connection, stored information must be encrypted with a user password.
- All personal data transfers must be ciphered (SSL channel at least, two layers of ciphering are better appreciated)

References

[1] SEAL D2.1 Technical Design of Cross-Sector Identities Linking Platform. 2019

[2] D3.2 Technical documentation on modular interfaces for different types of identities. 2020

[3] D3.3 Technical documentation on web and mobile user interfaces. 2020

[4] OpenAPI Specification, https://www.openapis.org/, retrieved 2020-05-19

APPENDIX 1 Deployment Guidelines

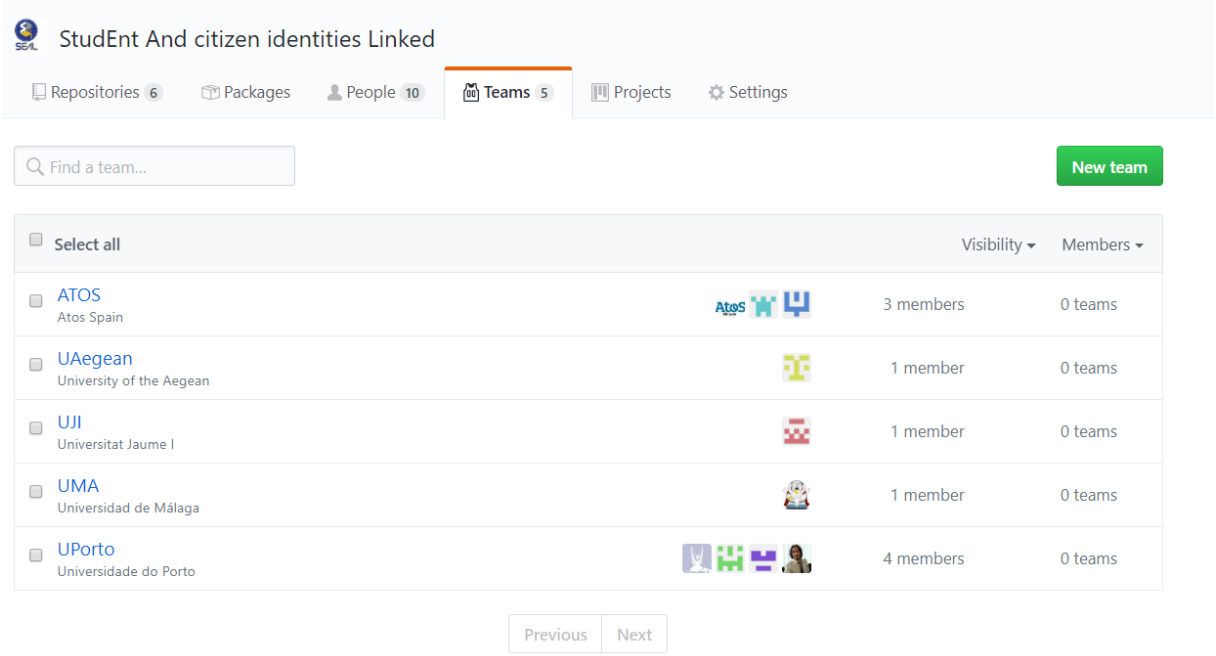
5.8 Introduction

5.9 Overview

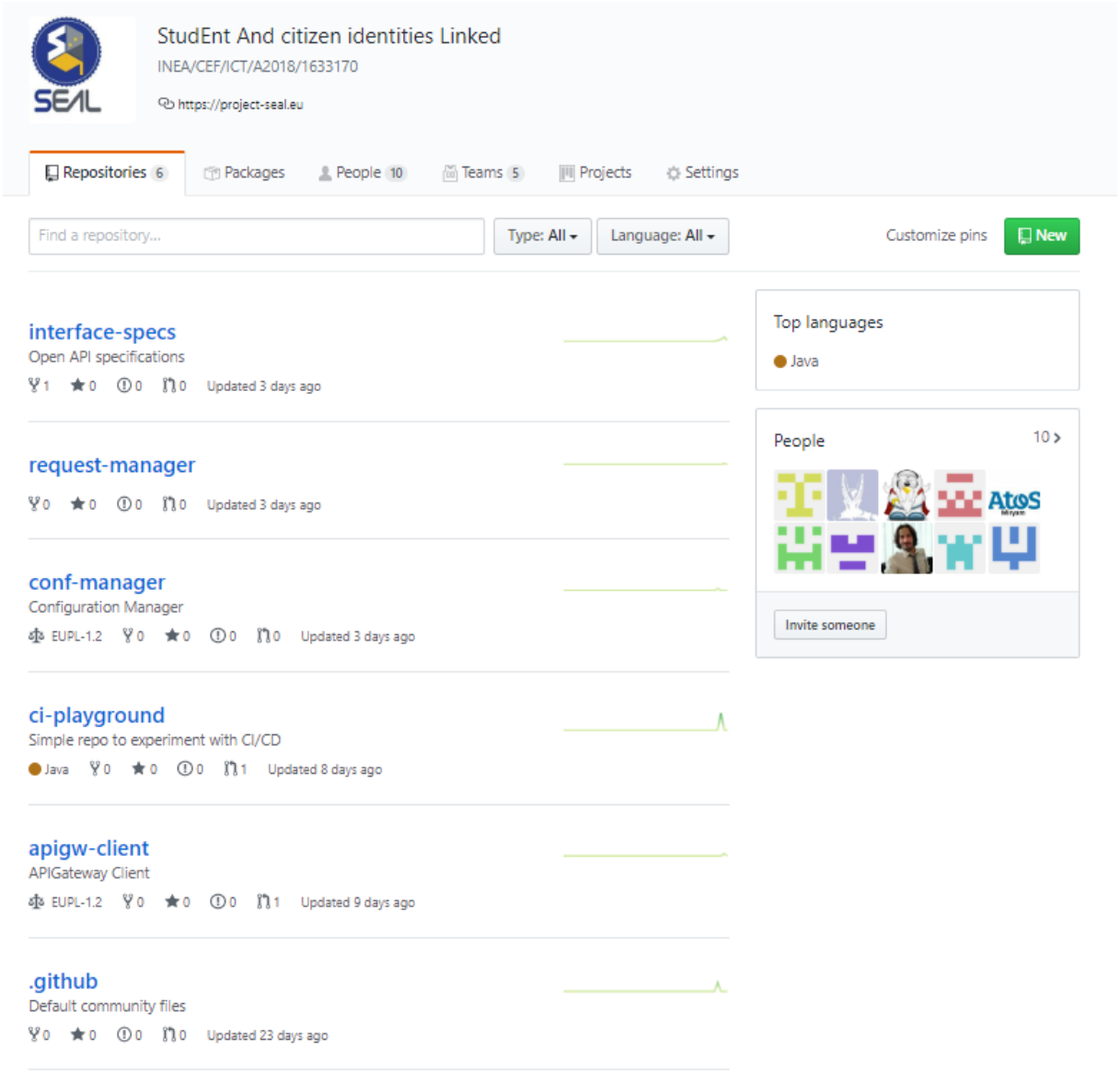
This document sets the guidelines for the CI/CD flow during the development of the SEAL project.

The aim of our CI/CD approach is to allow the SEAL partner design teams to work together more efficiently to quickly and automatically test, package, and deploy the SEAL software modules.

For that, the following github organization has been created: **EC-SEAL** organization with five teams with a maintainer, one per partner.



Note the repository "**.github**" under the EC-SEAL organization (https://github.com/EC-SEAL/.github), is used to define the way of participating in it.



5.10 Code of Conduct

A generic code of conduct for the project is found here: https://github.com/EC-SEAL/.github/blob/master/CODE_OF_CONDUCT.md

5.11 Contribution Guidelines

General information on pull requests, code conventions, commit conventions, language etc. is specified in https://github.com/EC-SEAL/.github/blob/master/CONTRIBUTING.md

5.12 Templates

Templates have been defined to help creating issues (https://github.com/EC-SEAL/.github/tree/master/.github/ISSUE_TEMPLATE) and pull requests (https://github.com/EC-SEAL/.github/blob/master/PULL_REQUEST_TEMPLATE.md).

Note the bug report, feature request and question templates included as issue templates.

The repositories are being added and their settings need to be tuned in order to follow the same basic lines through the Organization. As a minimum, "Collaborators & teams" and "Branches" options for each repository will be customized in a similar way, as explained below.

When a new repository is created, the following steps are recommended:

5.13 Select the License

We will use EUPL 1.2

See https://github.com/EC-SEAL/conf-manager/blob/development/LICENSE.md

A DISCLAIMER file and a LICENSE-AGREEMENT file (to check with the Legal Dept) are recommended for each repository (*TODO* an example in conf-manager repo).

README file

LET'S WRITE IT before starting to upload any code!!

Use of Headers

Every source file must have a header. This is an example for Atos code:

```
/**
Copyright © 2019  Atos Spain SA. All rights reserved.

This file is part of SEAL Configuration Manager (SEAL ConfMngr).
SEAL ConfMngr is free software: you can redistribute it and/or modify it under the terms of EUPL 1.2.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT,
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
See README file for the full disclaimer information and LICENSE file for full license
information in the project root.

@author Atos Research and Innovation, Atos SPAIN SA

<Short description of the class>
*/
```

5.14 Java package naming

Our proposal is to use “eu.SEAL.xxxxx”.

CI Flow

5.15 Guidelines

When a new repository is created, add team first, and then specify the settings for branches. For each repository, the default branch represents the "base" branch in that repository. All the pull requests and code commits are automatically made on it, unless you specify a different branch.

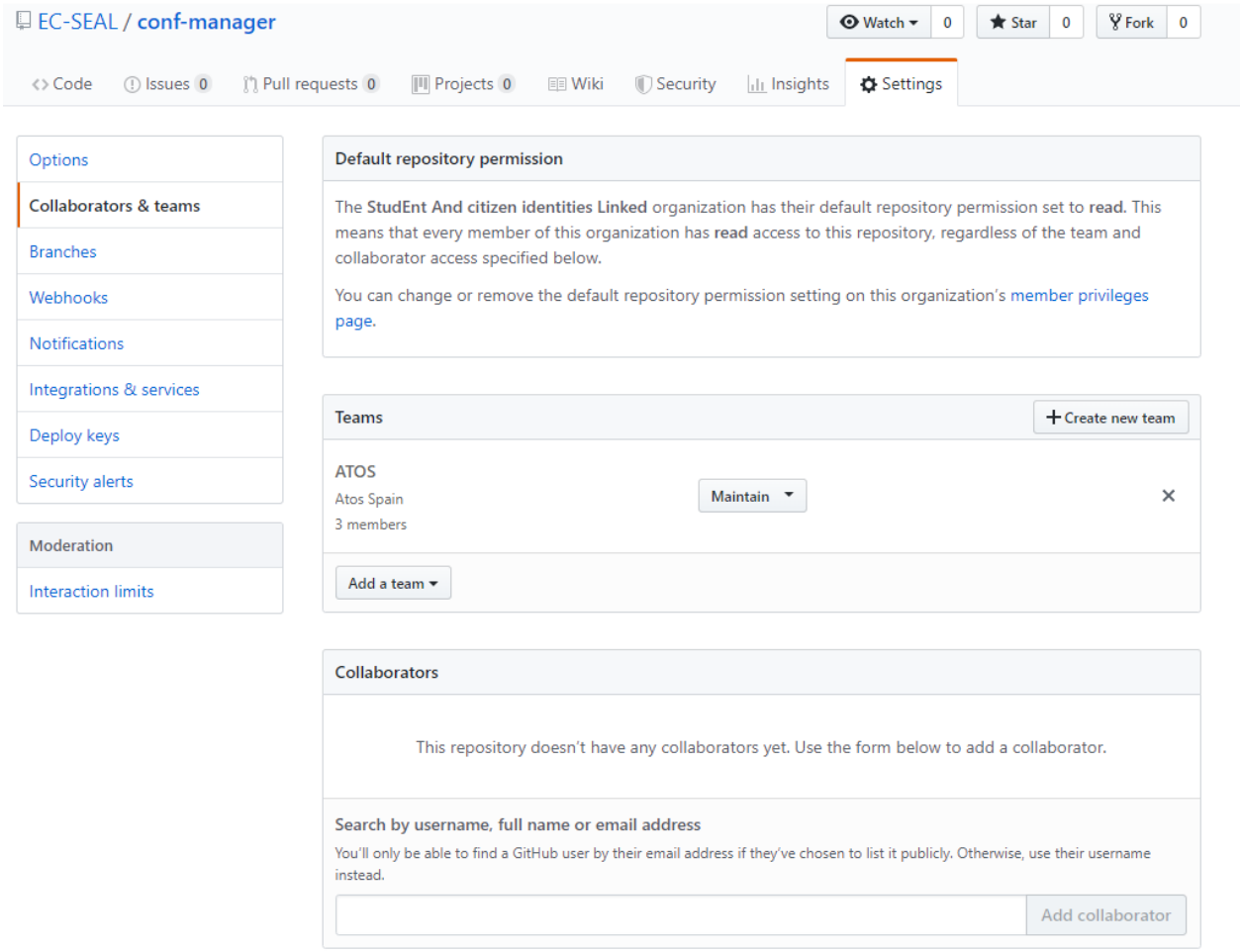
Remember to add a branch protection rule for these branches at least:

- master
- development

The settings to update would be:

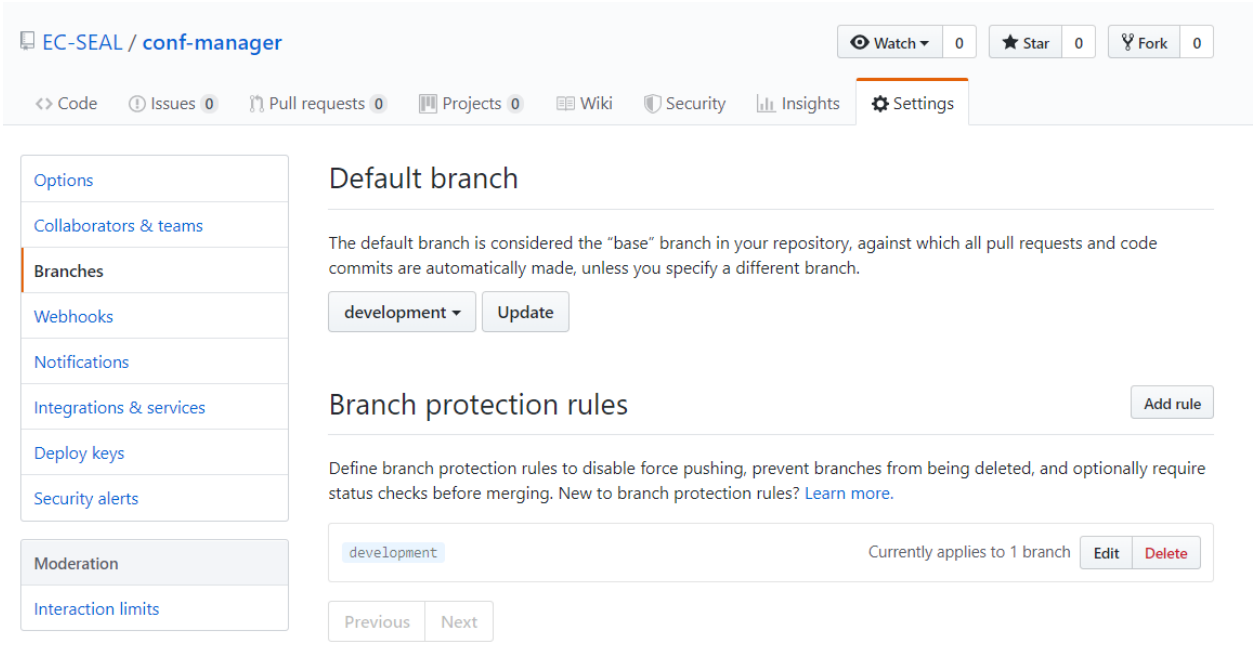
1. Require **pull request reviews** before merging to master (production environment) and development (pre or integration environment), i.e**., when a deployment is to take place later on**.
    - *TO EXPLAIN the different kind of critic branches.*
2. Require status checks to pass before merging
3. Require branches to be up to date before merging
4. Include administrators (not mandatory)
5. Restrict who can push to matching branches

See below, the conf-manager repository as an example, where ATOS team is the maintainer of the config-manager repo.

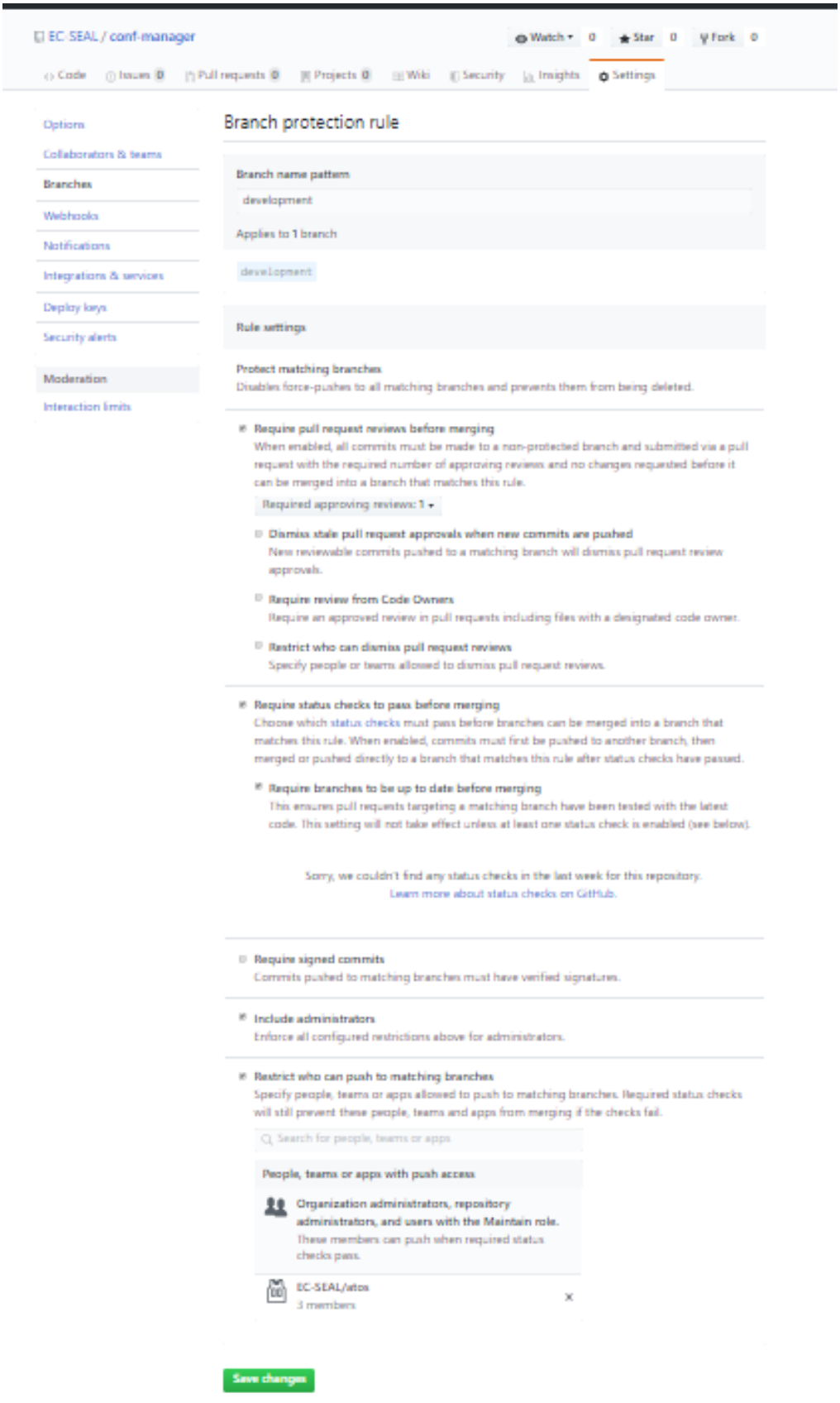


The objective is the master branches of all the SEAL repos to be aligned. For that, a **master** rule, the most restrictive is to be added to each repository. For the **development** branches (**default branch)**, minor restriction; *other* branches created, the less restrictive.

In this way, the default branch is the called "development" for the config-manager repo, as shown below:



And the following rule is defined for this branch:


EC-SEAL / conf-manager
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Security
Insights
Settings
Options
Collaborators & teams
Branches
Webhooks
Notifications
Integrations & services
Deploy keys
Security alerts
Moderation
Interaction limits
Branch protection rule
Branch name pattern
development
Applies to 1 branch
development
Rule settings
Protect matching branches
Disables force-pushes to all matching branches and prevents them from being deleted.
Require pull request reviews before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.
Required approving reviews: 1
Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
Require review from Code Owners
Require an approved review in pull requests including files with a designated code owner.
Restrict who can dismiss pull request reviews
Specify people or teams allowed to dismiss pull request reviews.
Require status checks to pass before merging
Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.
Require branches to be up to date before merging
This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).
Sorry, we couldn't find any status checks in the last week for this repository.
Learn more about status checks on GitHub.
Require signed commits
Commits pushed to matching branches must have verified signatures.
Include administrators
Enforce all configured restrictions above for administrators.
Restrict who can push to matching branches
Specify people, teams or apps allowed to push to matching branches. Required status checks will still prevent these people, teams and apps from merging if the checks fail.
Search for people, teams or apps
People, teams or apps with push access
Organization administrators, repository administrators, and users with the Maintain role.
These members can push when required status checks pass.
EC-SEAL/atos
3 members
Save changes
© 2019 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub Pricing API Training Blog About

Note in the above figure there is set one reviewer. In case of more than one reviewer was established, or a given reviewer was defined, the number of the pull request will be required for accepting the change.

As Travis CI for Open Source has access to EC-SEAL, every repository created in EC-SEAL will be audited. To check the repository status of a given repository in relation to travis tasks, www.travis-ci.org/EC-SEAL/your-repo.

To see an example, take a look at https://travis-ci.org/EC-SEAL/ci-playground/, and https://github.com/EC-SEAL/ci-playground/blob/development/.travis.yml to read the *travis file*.

It is recommended self-testing of the modules in the CI flow e.g. use unit tests in the pipeline such as JUnit can be used to easily mock dependencies.

5.16 SonarQube

*To be added here **SonarQube** to CI Flow in order to assess a good code quality.*

https://docs.travis-ci.com/user/sonarcloud/

2.3 Dockerhub

Read https://github.com/EC-SEAL/ci-playground/blob/development/.travis.yml, to see how the uploading to DockerHub is recommended to be automated.

CD flow

*To be added here the CD flow where **OpenShift** is a candidate tool to be used.*

5.18 Deployment in Atos Virtual Machine

By the moment, we are deploying the microservices in Atos VM (**vm.project-seal.eu**) by hand. That means the *execution* of the *docker-compose files* are done by the person who deploys the last change in her microservice.

Here it is shown an example of the docker file that deploys the Configuration Manager and the Session Manager. Note that the automatic procedure finished when the docker container was stored in the Docker Hub (https://hub.docker.com/).

```
version: '3'
volumes:
  # volume to store mysql database data
  data_sql:
services:
    Memcached:
      image: sameersbn/memcached:1.5.6
      ports:
        - 19211:11211
#      restart: always

    docker-mysql:
      image: mysql
#      command: --default-authentication-plugin=mysql_native_password
      #restart: always
      environment:
        MYSQL_ROOT_PASSWORD: example
        MYSQL_PASSWORD: example
        MYSQL_DATABASE: sessionMngr
      ports:
        - 9307:3306

    ConfManager:
        image: mvjatos/seal-cm:test
        environment:
            - KEYSTORE_PATH=/resources/testKeys/keystore.jks
            - KEY_PASS=selfsignedpass
            - STORE_PASS=keystorepass
            - HTTPSIG_CERT_ALIAS=1
            - SIGNING_SECRET=QjG+wP1CbAH2z4PWlWIDkxP4oRlgK2vos5/jXFfeBw8=
```

```
      - ASYNC_SIGNATURE=true
      - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
      - SSL_STORE_PASS=AtosCert1
      - SSL_KEY_PASS=AtosCert1
      - SSL_CERT_ALIAS=atoscert
    volumes:
      - /SEAL/CM/resources:/resources
    ports:
      - 9083:8083

  SessionManager:
    #image: endimion13/esmo-session-manager:0.1.5
    image: endimion13/esmo-session-manager:0.1.8a
    environment:
      - KEYSTORE_PATH=/resources/testKeys/keystore.jks
      - KEY_PASS=selfsignedpass
      - STORE_PASS=keystorepass
      - JWT_CERT_ALIAS=selfsigned
      - HTTPSIG_CERT_ALIAS=selfsigned
      - SIGNING_SECRET=QjG+wP1CbAH2z4PWlWIDkxP4oRlgK2vos5/jXFfeBw8=
      - ASYNC_SIGNATURE=true
      - EXPIRES=5
      - CONFIG_JSON=/resources/configurationResponse.json
      - CONFIGURATION_MANAGER_URL=https://vm.project-seal.eu:8083
      - MEMCACHED_HOST=Memcached
      - MEMCACHED_PORT=11211
      - DATABASE_HOST=docker-mysql
      - DATABASE_USER=root
      - DATABASE_PASSWORD=example
      - DATABASE_NAME=sessionMngr
      - DATABASE_PORT=3306
      #- ISSUER=SMms001

    volumes:
      - /SEAL/SM/resources:/resources
    links:
      - docker-mysql:mysql
      - ConfManager:vm.project-seal.eu
    ports:
      - 9090:8090
    depends_on:
      - docker-mysql
      - ConfManager
```

Remember the set of configuration files read by the **Configuration Manager**, which have to be available (see https://github.com/EC-SEAL/conf-manager/blob/development/README.md for more detail). In particular, it is very important the file msMetadataList.json, where the right endpoints of the microservices and the current authorizations are declared. Some examples are following:


```
[

  {
    "msId": "CMms001",
    "authorisedMicroservices": [
      "SMms001",
      "CLms001",
      "IDPms001",
        "APms001",
        "SAMLms_0001"
    ],
    "rsaPublicKeyBinary":
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCi7jZVwQFxQ2SY4lxjr05IexolQJJobwYzrvE5pk7Ac
QpG46kuJBzD8ziiqFFCGSNZ07cLWys+b5JmJ6kU44lKLVeGbEpgaO0OTBDLMk2fi5U83T8dezgWgaPFiy/
N3sHPpcW2Y3ZePo0UbM7MLzv14TR+jxTOyrmwWwGwDJsz+wIDAQAB",
    "publishedAPI": [
      {
        "apiClass": "CM",
        "apiCall": "microservices",
        "apiConnectionType": "get",
        "apiEndpoint": "https://vm.project-seal.eu:8083/cm/metadata/microservices"
      },
      {
        "apiClass": "CM",
        "apiCall": "externalEntities",
        "apiConnectionType": "get",
        "apiEndpoint":                                    "https://vm.project-
seal.eu:8083/cm/metadata/externalEntities"
      },
      {
        "apiClass": "CM",
        "apiCall": "attributes",
        "apiConnectionType": "get",
        "apiEndpoint": "https://vm.project-seal.eu:8083/cm/metadata/attributes"
      },
      {
        "apiClass": "CM",
        "apiCall": "internal",
        "apiConnectionType": "get",
```

```
        "apiEndpoint": "https://vm.project-seal.eu:8083/cm/metadata/internal"
      }
    ]
  },

  {
    "msId": "SMms001",
    "authorisedMicroservices": [
      "IDPms001",
      "CLms001",
      "SAMLms_0001",
       "APms001"
    ],
    "rsaPublicKeyBinary":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkvZf4Lm7dqp17tk/ICI+cCilI3yLfQraHy4px
FYDNn29l9eHnYRFnN9jBKKvOzSxf2zQkigNcHhIi96s7G4/xPL3rVaYepp/xfCKn5vkZeqg1PFOE0HqDKC
nIbLxNdnHYDLICQrd1PRTdFHnwRpLouF6B3PCZpQL5XxX3WFzg2KZ2U1NIdVLJjWb3AY1SJ4kIYAOIwn6A
ZQPum4i5G4M9QQj3KGl164007TUx27rxzBVILpm+knxYjUiipqZ/5kiDdTxYBPR0qDVIhSl3hk9RhSI95s
7unrll8rb3E8w1ORrfTQNg1UlpGgww3jZi3GLScLEK3ghwg5H5gL/2SSiEwIDAQAB",
    "publishedAPI": [
      {
        "apiClass": "SM",
        "apiCall": "endSession",
        "apiConnectionType": "post",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/endSession"
      },
      {
        "apiClass": "SM",
        "apiCall": "generateToken",
        "apiConnectionType": "get",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/generateToken"
      },
      {
        "apiClass": "SM",
        "apiCall": "getSession",
        "apiConnectionType": "get",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/getSession"
      },
      {
        "apiClass": "SM",
        "apiCall": "getSessionData",
        "apiConnectionType": "get",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/getSessionData"
      },
```

```
      {
        "apiClass": "SM",
        "apiCall": "startSession",
        "apiConnectionType": "post",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/startSession"
      },
      {
        "apiClass": "SM",
        "apiCall": "updateSessionData",
        "apiConnectionType": "post",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/updateSessionData"
      },
      {
        "apiClass": "SM",
        "apiCall": "validateToken",
        "apiConnectionType": "get",
        "apiEndpoint": "http://vm.project-seal.eu:8090/sm/validateToken"
      }
    ]
  },
  {
    "msId": "CLms001",
    "authorisedMicroservices": [
      "IdPms001"
    ],
    "rsaPublicKeyBinary":
"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCi7jZVwQFxQ2SY4lxjr05IexolQJJobwYzrvE5pk7Ac
QpG46kuJBzD8ziiqFFCGSNZ07cLWys+b5JmJ6kU44lKLVeGbEpgaO0OTBDLMk2fi5U83T8dezgWgaPFiy/
N3sHPpcW2Y3ZePo0UbM7MLzv14TR+jxTOyrmwWwGwDJsz+wIDAQAB",
    "publishedAPI": [
      {
        "apiClass": "CL",
        "apiCall": "clCallback",
        "apiConnectionType": "post",
        "apiEndpoint": "https://vm.project-seal.eu:8053/cl/callback"
      }

    ]
  },
…
```

APPENDIX 2 Docker Compose Example

```
version: '3'
volumes:
  # volume to store mysql database data
  data_sql:
services:
    Memcached:
      image: sameersbn/memcached:1.5.6
      ports:
        - 19211:11211
#       restart: always

    docker-mysql:
      image: mysql
#       command: --default-authentication-plugin=mysql_native_password
      #restart: always
      environment:
        MYSQL_ROOT_PASSWORD: xxxxx
        MYSQL_PASSWORD: xxxxx
        MYSQL_DATABASE: sessionMngr
      ports:
        - 9307:3306

    ConfManager:
        image: mvjatos/seal-cm:0.0.3
        environment:
            - KEYSTORE_PATH=/resources/testKeys/keystore.jks
            - KEY_PASS=xxxxx
            - STORE_PASS=xxxxx
            - HTTPSIG_CERT_ALIAS=xxxxx
            - SIGNING_SECRET=xxxxx
            - ASYNC_SIGNATURE=true
            - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
            - SSL_STORE_PASS=xxxxx
            - SSL_KEY_PASS=xxxxx
            - SSL_CERT_ALIAS=xxxxx
        volumes:
            - /SEAL/CM/resources:/resources
        ports:
          - 9083:8083

    SessionManager:
        image: endimion13/esmo-session-manager:0.1.8a
        environment:
```

```
        - KEYSTORE_PATH=/resources/testKeys/keystore.jks
        - KEY_PASS=xxxxx
        - STORE_PASS=xxxxx
        - JWT_CERT_ALIAS=xxxxx
        - HTTPSIG_CERT_ALIAS=xxxxx
        - SIGNING_SECRET=xxxxx
        - ASYNC_SIGNATURE=true
        - EXPIRES=5
        - CONFIG_JSON=/resources/configurationResponse.json
        - CONFIGURATION_MANAGER_URL=https://vm.project-seal.eu:8083
        - MEMCACHED_HOST=Memcached
        - MEMCACHED_PORT=11211
        - DATABASE_HOST=docker-mysql
        - DATABASE_USER=root
        - DATABASE_PASSWORD=xxxxx
        - DATABASE_NAME=sessionMngr
        - DATABASE_PORT=3306

    volumes:
        - /SEAL/SM/resources:/resources
    links:
        - docker-mysql:mysql
        - ConfManager:vm.project-seal.eu
    ports:
      - 9090:8090
    depends_on:
      - docker-mysql
      - ConfManager

emrtd-is:
    image: mvjatos/seal-emrtd-is:0.0.1
    environment:
        - KEYSTORE_PATH=/resources/testKeys/keystore.jks
        - KEY_PASS=xxxxx
        - STORE_PASS=xxxxx
        - HTTPSIG_CERT_ALIAS=xxxxx
        - SIGNING_SECRET=xxxxx
        - ASYNC_SIGNATURE=true
        - CONFIGURATION_MANAGER_URL=https://vm.project-seal.eu:8083
        - SESSION_MANAGER_URL=http://SessionManager:8090
        - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
        - SSL_STORE_PASS=xxxxx
        - SSL_KEY_PASS=xxxxx
        - SSL_CERT_ALIAS=xxxxx
    volumes:
        - /SEAL/EMRTD-IS/resources:/resources
    links:
```

```
        - ConfManager:vm.project-seal.eu
        - SessionManager:SessionManager
    ports:
      - 9073:8073
    depends_on:
      - SessionManager
      - ConfManager

apigwcl:
    image: mvjatos/seal-apigwcl:test
    environment:
        - KEYSTORE_PATH=/resources/testKeys/keystore.jks
        - KEY_PASS=xxxxx
        - STORE_PASS=xxxxx
        - HTTPSIG_CERT_ALIAS=xxxxx
        - SIGNING_SECRET=xxxxx
        - ASYNC_SIGNATURE=true
        - CONFIGURATION_MANAGER_URL=https://vm.project-seal.eu:8083
        - SESSION_MANAGER_URL=http://SessionManager:8090
        - SSL_KEYSTORE_PATH=/resources/keystoreatos.jks
        - SSL_STORE_PASS=xxxxx
        - SSL_KEY_PASS=xxxxx
        - SSL_CERT_ALIAS=xxxxx
    volumes:
        - /SEAL/APIGWCL/resources:/resources
    links:
        - ConfManager:vm.project-seal.eu
        - SessionManager:SessionManager
    ports:
      - 9053:8053
    depends_on:
      - SessionManager
      - ConfManager

edugain-idp:
    image: cbuendiaatos/idp-edugain:apigw-test
    restart: on-failure
    command:
      - "-Dorg.opensaml.httpclient.https.disableHostnameVerification=true"
    environment:
      - ASYNC_SIGNATURE=true
      - KEY_PASS=xxxxx
      - SIGNING_SECRET=xxxxx
      - JWT_CERT_ALIAS=xxxxx
      - SAML_KEYSTORE_PATH=classpath:/saml/your_site_name.jks
      - SAML_KEYSTORE_PASS=xxxxx
      - SAML_KEYSTORE_ID=xxxxx
```

```
      - SAML_KEY_PASS=xxxxx
      - STORE_PASS=xxxxx
      - HTTPSIG_CERT_ALIAS=xxxxx
      - SESSION_MANAGER_URL=http://SessionManager:8090
      - KEYSTORE_PATH=resources/testKeys/keystore.jks
      - IDP_METADATA_URL=https://eid-proxy.aai-dev.grnet.gr/Saml2IDP/proxy.xml
      #- YES: IDP_METADATA_URL=https://idp.ssocircle.com
      - KEYSTORE_PASS=xxxxx
      - KEYSTORE_ID=apollo
      - TESTING=true
    links:
        - SessionManager:SessionManager
    external_links:
        - kc_keycloak_1:vm.project-seal.eu
    networks:
        - default
        - kc_default
    volumes:
      - ./EDUGAIN-IDP/resources:/resources
    ports:
      - 10081:8080
    depends_on:
      - SessionManager


IdPMS:
    image: endimion13/seal-eidas-idp:0.0.7n
    environment:
        - EIDAS_PROPERTIES=CurrentFamilyName,CurrentGivenName,DateOfBirth,PersonIdentifier
        - SP_COUNTRY=GR
        - SP_SECRET=xxxxx
        - AUTH_DURATION=1800
        - SESSION_MANAGER_URL=http://SessionManager:8090
        - KEY_PASS=xxxxx
        - JWT_CERT_ALIAS=xxxxx
        - ASYNC_SIGNATURE=true
        - HTTPSIG_CERT_ALIAS=xxxxx
        - KEYSTORE_PATH=resources/testKeys/keystore.jks

        - STORE_PASS=xxxxx
        - SEAL_SUPPORTED_SIG_ALGORITHMS=RSA
        - SEAL_SUPPORTED_ENC_ALGORITHMS=RSA
        - RESPONSE_SENDER_ID=eIDAS-IdP
        - RESPONSE_RECEIVER=CLms001 #API_GW
        - SEAL_EXPOSE_URL=/as/authenticate , /is/query
        - SEAL_ENTITY_ID=http://vm.project-seal.eu:8091/eidas-idp
        - SEAL_DEFAULT_NAME=SEAL_EIDAS_IDP
```

```
        volumes:
            - /SEAL/EIDAS-IDP/resources:/resources
        ports:
          - 8091:8080
          #- 8443:8443

    Persistence:
        image: bdpereira/ec-seal-per:perseal
        environment:

            - AUTH_URL=https://login.live.com/oauth20_authorize.srf
            - REDIRECT_URL=http://vm.project-seal.eu:9082/per/code
            - REDIRECT_URL_HTTPS=https://vm.project-seal.eu:9082/per/code
            - FETCH_TOKEN_URL=https://login.microsoftonline.com/common/oauth2/v2.0/token
            - CREATE_FOLDER_URL=https://graph.microsoft.com/v1.0/me/drive/root/children
            - CREATE_FILE_URL=https://graph.microsoft.com/v1.0/me/drive/items/
            - GET_FOLDER_URL=https://graph.microsoft.com/v1.0/me/drive/root
            - SM_ENDPOINT=http://SessionManager:8090/sm
            - KEYSTORE=resources/keystore.jks
            - ALIAS=xxxxx
            - KEYSTOREPASS=xxxxx
            - INTER=inter.p12

            - HOST=Persistence:8082

            - GOOGLE_DRIVE=googleDrive
            - ONE_DRIVE=oneDrive
            - DATA_STORE_FILENAME=datastore.txt
            - PASS=qwerty

            - GOOGLE_DRIVE_CLIENT_ID=425112724933-
9o8u2rk49pfurq9qo49903lukp53tbi5.apps.googleusercontent.com
            - GOOGLE_DRIVE_CLIENT_PROJECT=seal-274215
            - GOOGLE_DRIVE_AUTH_URI=https://accounts.google.com/o/oauth2/auth
            - GOOGLE_DRIVE_TOKEN_URI=https://oauth2.googleapis.com/token
            - GOOGLE_DRIVE_AUTH_PROVIDER=https://www.googleapis.com/oauth2/v1/certs
            - GOOGLE_DRIVE_CLIENT_SECRET=0b3WtqfasYfWDmk31xa8UAht
            - GOOGLE_DRIVE_REDIRECT_URIS=http://vm.project-
seal.eu:9082/per/code,http://perseal.seal.eu:8082/per/code,https://localhost:8082/per/code
            - ONE_DRIVE_CLIENT_ID=fff1cba9-7597-479d-b653-fd96c5d56b43
            - ONE_DRIVE_SCOPES=offline_access files.read files.read.all files.readwrite
files.readwrite.all

            - PERSEAL_INT_PORT=8082
            - PERSEAL_EXT_PORT=8082

        command: /go/bin/perseal
```

```
    volumes:
        - /SEAL/PER/resources:/resources

    links:
        - SessionManager:SessionManager
    ports:
      - 8082:8082
    depends_on:
      - SessionManager

reconciliation:
    image: faragom/seal-reconciliation:b
    environment:
        - ENCRYPTION_KEY=sealEncryptionKey
        - PROPERTIES_FILE=/app/data/server.properties
        - MSPORT=8050  # Internal port
        - WTHREADS=4

    volumes:
        - /SEAL/RECONCILIATION/app/data:/app/data
    links:
        - ConfManager:vm.project-seal.eu
        - ConfManager:confmanager
        - SessionManager:sessionmanager
        - SessionManager:SessionManager
    ports:
      - 9050:8050
    depends_on:
      - ConfManager

uportissuer:
    image: endimion13/seal-issuer:0.0.3m
    environment:
      NODE_ENV: "production"
      ENDPOINT: http://vm.project-seal.eu:4000
      HTTPS_COOKIES: "true"
      SENDER_ID: "IdPms001"
      RECEIVER_ID: "IdPms001"

      MEMCACHED_URL: Memcached:11211

      ## optional if not set default values are used
      SEAL_SM_URI : SessionManager #defaults to 'vm.project-seal.eu'
      SEAL_SM_PORT: 8090  #defaults to '9090'
      SEAL_EDUGAIN_PORT: 10081 #defaults to ''

    links:
```

```
            - SessionManager:SessionManager


        ports:
          - 4000:3000


networks:
    kc_default:
      external: true
```